MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. Memo No. 605 December, 1980

Spatial Planning: A Configuration Space Approach

Tomás Lozano-Pérez

ABSTRACT. This paper presents algorithms for computing constraints on the position of an object due to the presence of obstacles. This problem arises in applications which require choosing how to arrange or move objects among other objects. The basis of the approach presented here is to characterize the position and orientation of the object of interest as a single point in a Configuration Space, in which each coordinate represents a degree of freedom in the position and/or orientation of the object. The configurations forbidden to this object, due to the presence of obstacles, can then be characterized as regions in the Configuration Space. The paper presents algorithms for computing these Configuration Space obstacles when the objects and obstacles are polygons or polyhedra. An approximation technique for high-dimensional Configuration Space obstacles, based on projections of obstacles slices, is described.

Acknowledgements. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Office of Naval Research under Office of Naval Research contract N00014-77C-0389.

1. Introduction

An increasing range of computer applications deal with models of two- and three-dimensional objects. In these applications, objects must often be placed among other objects or moved in such a way so that no interference from nearby objects result. In this paper, these types of problems are called *spatial planning* problems. Among the many applications where spatial planning plays an important role are:

1. Planning the layout of a building [8], i.e. the arrangement of walls, corridors, rooms, and equipment so as to fulfill a user's design constraints as well as implicit consistency constraints.

2. Planning how to machine a part using a Numerically Controlled Machine Tool [47], which requires plotting the path of one or more cutting surfaces so as to produce the desired part.

3. Planning the layout of an IC chip [45] so as to minimize area, subject to geometric design constraints.

4. Planning how to assemble a part using an industrial robot [18] [19] [38], which requires choosing how to grasp objects, move them without collisions and bring them into contact.

Problems in spatial planning generally involve (at least) two important types of considerations:

1. Geometric — The legal solutions must be characterized, which involves considering interactions between the shapes of objects and obstacles.

2. Optimization — The best solution must be chosen from among the legal solutions.

This paper deals primarily with computing constraints on the position of an object due to the presence of obstacles, thus its focus is on the geometric aspect of spatial planning. The development throughout will be based on polyhedral object models, although many of the results and approaches are applicable to other classes of object models.

Recently, there has been a rapid growth of interest in efficient algorithms for geometric problems. Previous work[1] has focused on algorithms for (1) computing convex hulls [9] [12] [15] [29], (2) intersecting convex polygons and polyhedra [5] [24] [33] [35], (3) intersecting half-spaces [7] [30] (4) decomposing polygons [32], and (5) closest-point problems [34]. This paper formulates two other geometric

[1]The references cited here are representative of the current literature; they are by no means a complete survey.

problems, Findspace and Findpath, with important applications in spatial planning, describes an approach to their solution and presents algorithms for the central problem posed by the approach. Previous work on these problems is briefly reviewed in Section 11.

2. Spatial Planning Problems

Let R be a convex polyhedron that contains k_B other, possibly overlapping, convex polyhedra B_j designated as *obstacles*. Let A be the union of k_A convex polyhedra A_i, i.e. $A = \bigcup_{i=1}^{k_A} A_i$. Figure 1 illustrates two related geometric problems, defined below (where position is used to mean both translation and orientation):

1. **Findspace** — Find a position for A, inside R, such that $\forall i \forall j : A_i \cap B_j = \emptyset$. This is called a *safe* position.

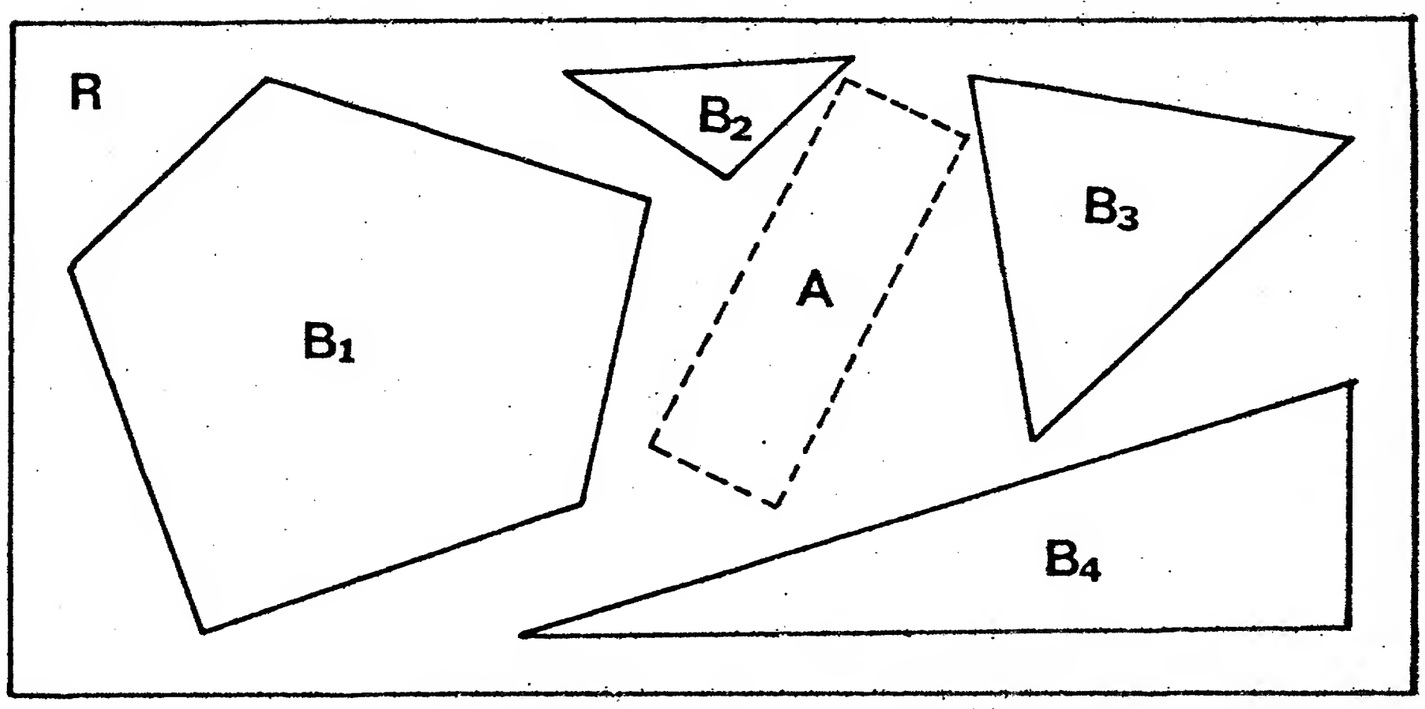
2. **Findpath** — Find a path for A from position s to position g such that A is always in R and A never overlaps any of the B_i. This is called a *safe* path.

Versions of the Findpath and Findspace problems occur in many spatial planning applications. For example, the approaches to object layout (template packing) in [1] [2] [3] and [10] are based on solutions to Findspace in two-dimensions. Also, systems for programming industrial robots using object models [20] [38] must solve 3-dimensional Findspace and Findpath problems when choosing grasp points on objects or paths for the robot.

3. The *Cspace* Approach to Spatial Planning

In this section, an overview of the Configuration Space approach to spatial planning will be presented. Sections 5 through 10 discuss the approach more formally.

The position and orientation of a rigid solid can be represented as a single 6-dimensional point, called its *configuration*. The 6-dimensional space of configurations for a solid, A, is called its Configuration Space and denoted $Cspace_A$. For example, a configuration may have one coordinate value for each of the x, y, z coordinates of a selected point on the object and one coordinate value for



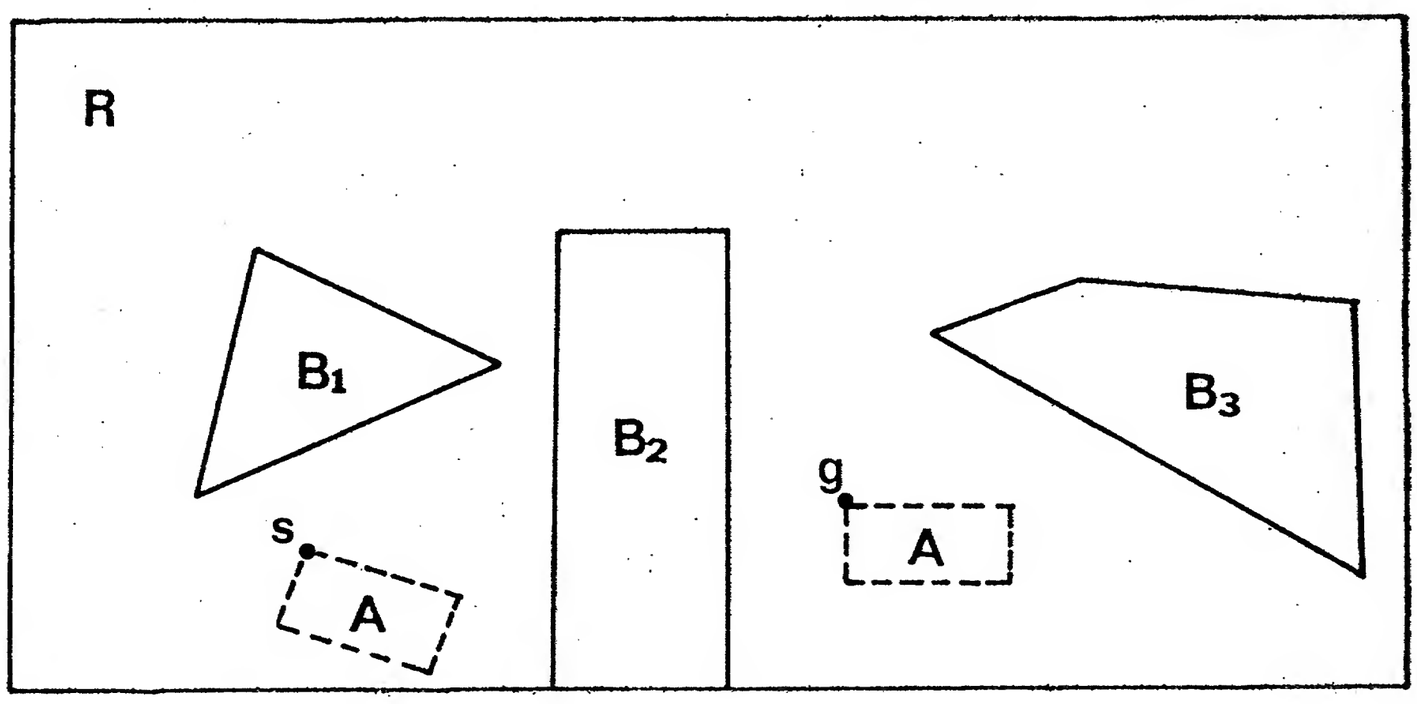


Figure 1. The definition of R, B_j and A for Findspace and Findpath problems in two dimensions. (a) The Findspace problem is to find a position for A which does not overlap any of the B_j. (b) The Findspace problem is to find a path for A from s to g that avoids the B_j.

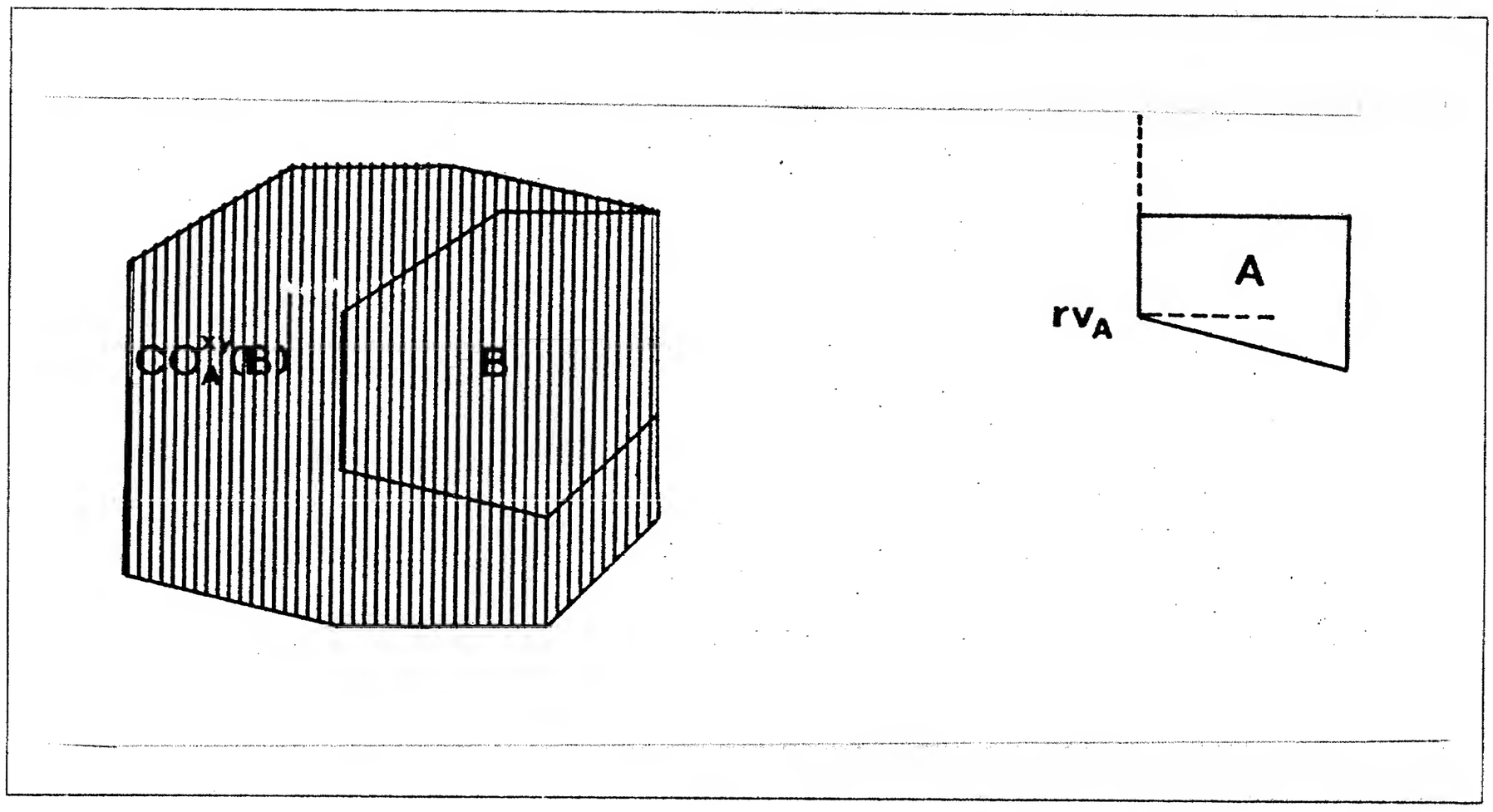


Figure 2. The $Cspace_A$ obstacle due to B, for fixed orientation of A.

each of the object's Euler angles[2] . In general, an n-dimensional configuration space can be used to model any system that can be characterized with n parameters. An example is the configuration of an industrial robot with n joints, where n is typically 5 or 6. In $Cspace_A$, the set of configurations where A *overlaps* B will be denoted $CO_A(B)$, the $Cspace_A$ *Obstacle* due to B. Similarly, those configurations where A is completely *inside* B will be denoted $CI_A(B)$, the $Cspace_A$ *Interior* of B. Together, these two $Cspace_A$ constructs embody all the information needed to solve Findspace and Findpath problems.

If the orientation of a convex polygon A is fixed, $Cspace_A$ is simply the (x, y) plane. This is so because the (x, y) position of some *reference vertex* rv_A is sufficent to specify the polygon's configuration. In this case, the presence of another convex polygon B constrains rv_A to be outside of $CO_A(B)$, a larger convex polygon, shown as the shaded region in Figure 2. Thus, the Findspace problem can be transformed to the equivalent problem of placing rv_A outside of $CO_A(B)$, but inside $CI_A(R)$. Similarly, for multiple obstacles B_i, a location for A is safe iff rv_A is not inside any of the

[2]The relative rotation of one coordinate system relative to another can be specified in terms of three angles usually referred to as Euler angles [43]. These angles indicate the magnitude of three successive rotations about specified axes, but no uniform convention for the choice of axes exists.

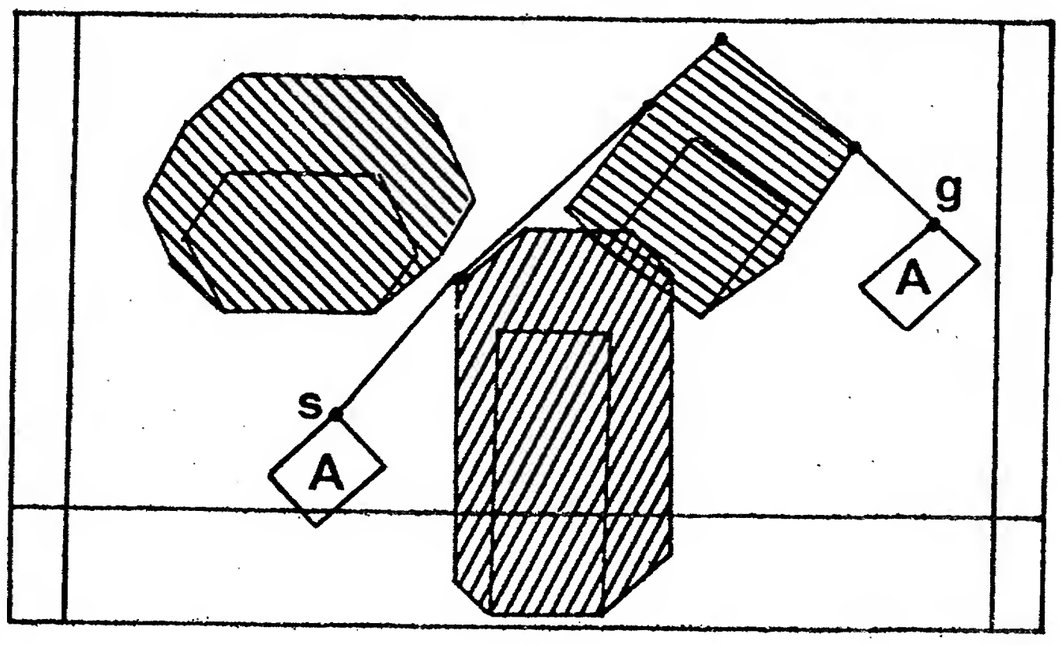


Figure 3. The Findpath problem and its formulation using the $CO_A^{xy}(B_j)$. The shortest collision-free paths connect the origin and the destination via the vertices of the CO polygons.

$CO_A(B_i)$, but is inside $CI_A(R)$. Subsequent sections discuss algorithms for $CO_A(B)$ and $CI_A(B)$.

If the orientation of A is fixed, then the Findpath problem for the polygon A among the B_i is equivalent to the Findpath problem for the point rv_A among the $CO_A(B_i)$. When the $CO_A(B_i)$ are polygons, the shortest[3] safe paths for rv_A are piecewise linear paths connecting the start and the goal via the vertices of the CO polygons, Figure 3. Therefore, Findpath can be formulated as a graph search problem. The graph is formed by connecting all pairs of CO vertices (and the start and goal) that can "see" each other, i.e. can be connected by a straight line that does not intersect any of the obstacles. The shortest path from the start to the goal in this *visibility graph* **(Vgraph)** is the shortest safe path for A among the B_i [21]. This algorithm efficiently solves Findpath problems when the orientation of A is fixed, but the paths it finds are very susceptible to inaccuracies in the object model. These paths touch the $Cspace_A$ obstacles, therefore if the model were exact, an object moving along this type of path would just touch the obstacles. But an inaccurate model may result in a collision. Furthermore, the Vgraph algorithm with three-dimensional objects and obstacles does not find optimal paths.

[3]This assumes Euclidean distance as a metric. For the optimality conditions using a rectilinear (Manhattan) metric, see [16].

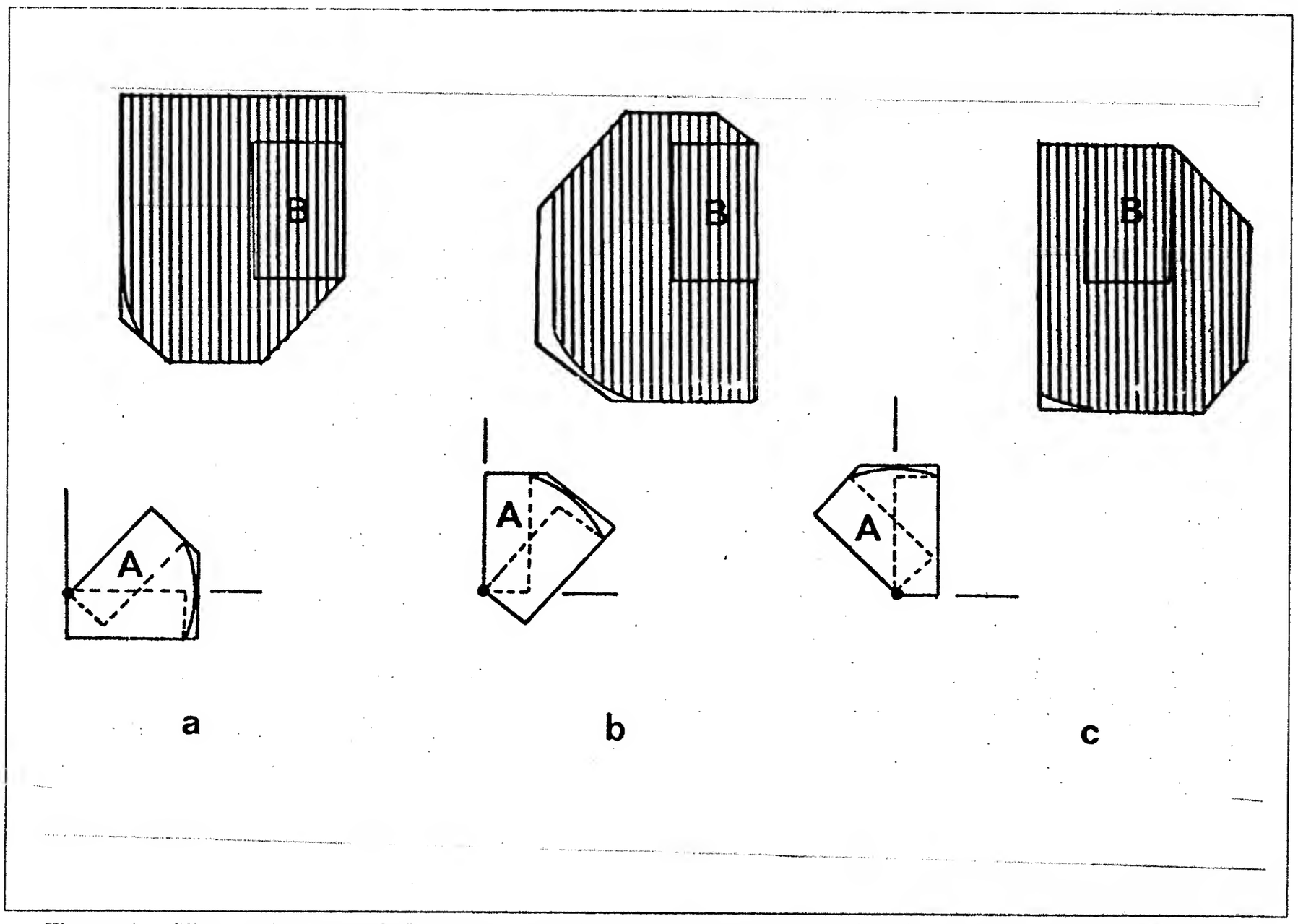


Figure 4. Slice projections of $Cspace_A$ obstacles computed using the (x, y)-area swept out by A over a range of θ values. Each of the shaded obstacles is the (x, y)-projection of a θ-slice of $CO_A(B)$. The figure also shows a polygonal approximation to the slice projection and the polygonal approximation to the swept volume from which it derives.

When A is a three-dimensional solid which is allowed to rotate, $CO_A(B)$ is a complicated curved object in a 6-dimensional $Cspace_A$. Rather than compute these objects directly, the approach taken here is to use a sequence of two- and three-dimensional projections of the high-dimensional $Cspace_A$ obstacles. In particular, the 6-dimensional $Cspace_A$ obstacles for a rigid solid can be approximated by several 3-dimensional projections of CO slices. A j-slice of an object $C \in \Re^n$ is defined to be $\{(\beta_1, \ldots, \beta_n) \in C \mid \gamma_j \leq \beta_j \leq \gamma'_j\}$, where γ_j and γ'_j are the lower and upper bounds of the slice, respectively. Then, if K is a set of indices between 1 and n, a K-slice is the intersection of all the j-slices for $j \in K$. Notice that a K-slice of C is an object of the same dimension as C. Slices can then be *projected* onto those coordinates not in K to obtain objects of lower dimension.

As an example of the slice projection technique, Figure 4 shows, shaded, the (x, y) projection of θ-slices of $CO_A(B)$ when A and B are convex polygons. These slices represent configurations where

A overlaps B for some orientation of A in the specified range of θ. Section 10 shows that these slice projections are the $Cspace_A$ obstacles of the area (volume) swept out by A over the range of orientations of the slice. Note that approximating the swept volume as a polyhedron leads to a polyhedral approximation for the projected slices, as shown in Figure 4.

The slice projection technique has two important properties:

1. A solution to a Findspace problem in any of the slices is a solution to the original problem, but since the slices are an approximation to the *Cspace* obstacle, the converse is not necessarily true.

2. The slice projection of $Cspace_A$ obstacle can be computed without having to compute the high-dimensional $Cspace_A$ obstacle, see Section 10.

The slice projection method can also be used to extend the **Vgraph** algorithm described earlier to find safe (but sub-optimal) paths when rotations of A are allowed [21]. A number of slice projections of the *Cspace* obstacles are constructed for different ranges of orientations of A. The problem of planning safe paths in the high-dimensional $Cspace_A$ is decomposed into (a) planning safe paths via CO vertices within each slice projection and (b) moving between slices, at configurations that are safe in both slices. Both of these types of motions can be modelled as links in the Vgraph, therefore the complete algorithm can be formulated as a graph search problem. This approach is illustrated in Figure 5. However, the Vgraph algorithm has several drawbacks when the obstacles are 3-dimensional. In particular,

1. Optimal paths in higher dimensions do not typically traverse the vertices of the *Cspace* obstacles.

2. In higher dimensions, there may be *no* paths via vertices, within the enclosing polyhedral region R, although other types of safe paths within R may exist.

These drawbacks may be alleviated by introducing additional nodes in the Vgraph which do not correspond to vertices [21]. An alternative strategy to finding safe paths in $Cspace_A$ is discussed in [20].

The key to the *Cspace* approach outlined above is computing the *Cspace* obstacles; the rest of the paper is devoted to this problem.

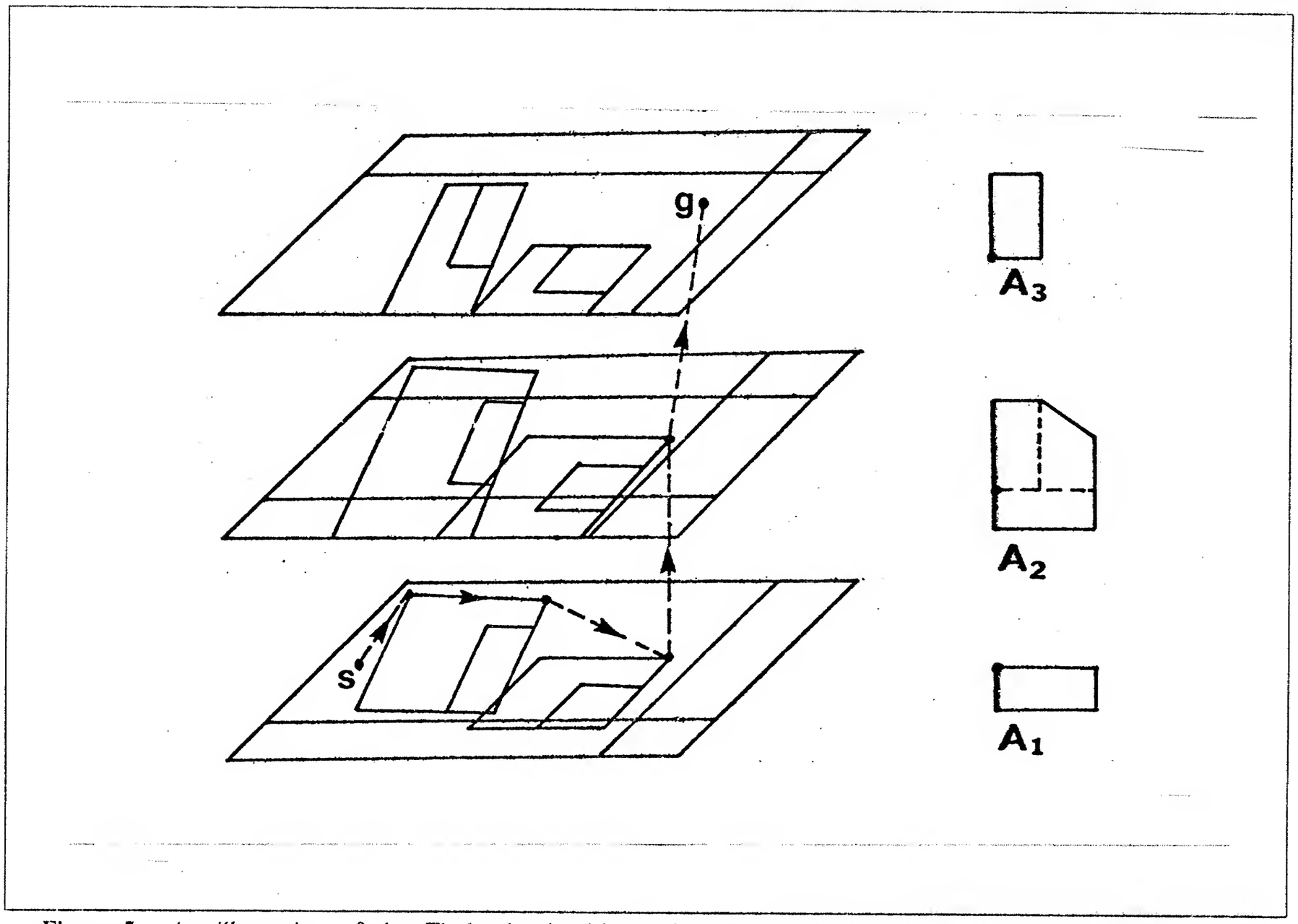


Figure 5. An illustration of the Findpath algorithm using slice projection described by Lozano-Perez and Wesley in [21]. A number of slice projections of the *Cspace* obstacles are constructed for different ranges of orientations of A. The problem of planning safe paths in the high-dimensional $Cspace_A$ is decomposed into (1) planning safe paths via CO vertices within each slice projection and (2) moving between slices, at configurations that are safe in both slices. A_1 represents A in its initial configuration. A_3 represents A in its final configuration and A_2 is a simple polyhedral approximation to the swept volume of A between its initial and final orientation.

4. Notation and Conventions

All geometric entities — points, lines, edges, planes, faces and objects — will be treated as (infinite) sets of points. All of these entities will be in $\Re^k$, the k-dimensional real Euclidean space. a, b, x, and y shall denote points of $\Re^k$, as well as the corresponding vectors. A, B, and C shall denote sets of points in $\Re^k$, while I and K shall denote sets of integers. γ, θ, and β, shall denote reals, while n, i, j, k shall be used for integers. The coordinate representation of a point $c \in \Re^n$, for any n, shall be $c = (\gamma_i) = (\gamma_1, \ldots, \gamma_n)$. The magnitude of a vector a will be $\|a\|$ and the cardinality of a set A will be $|A|$. The *scalar (dot) product* of vectors a and b will be denoted $\langle a, b \rangle$.

The following operations are defined on sets of points in $\Re^n$:

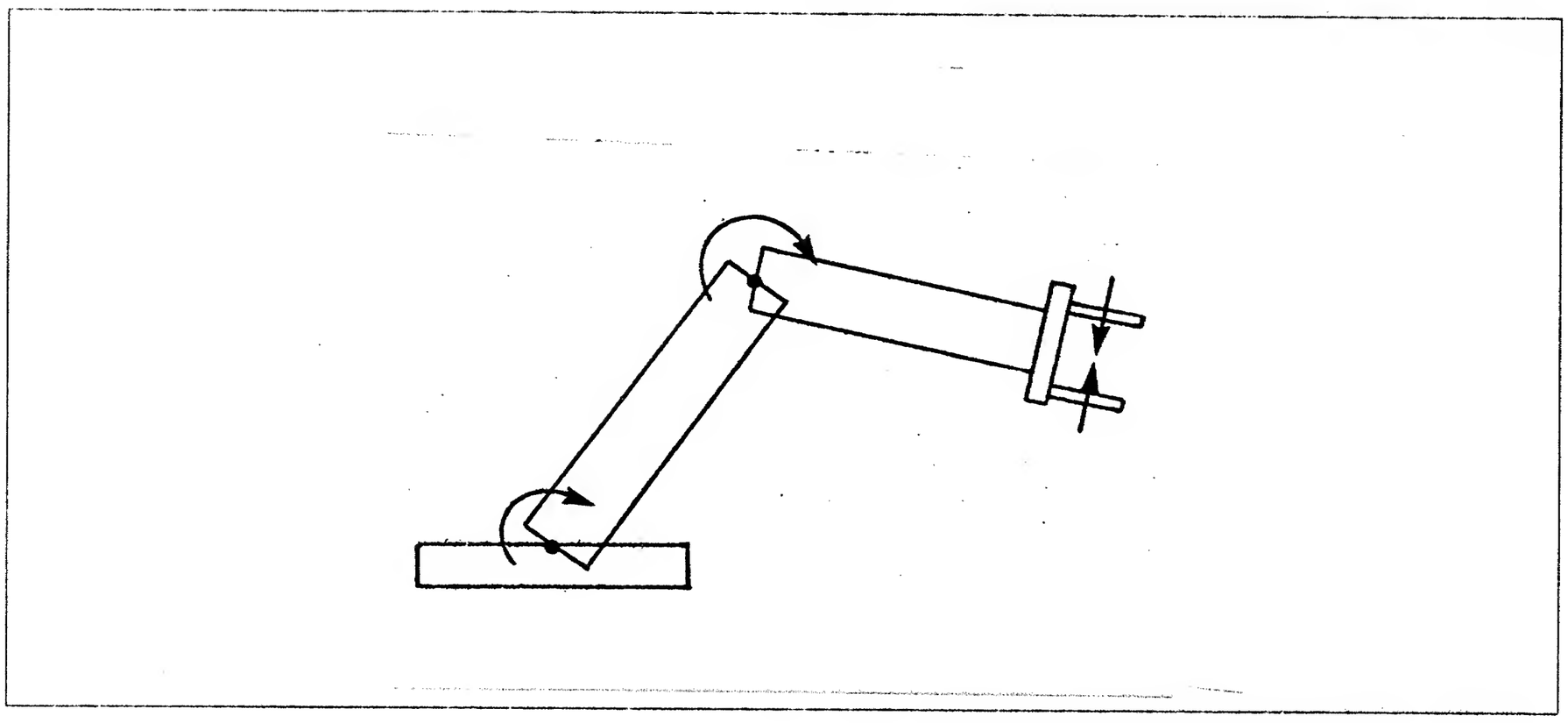


Figure 6. Linked Polyhedra can be used to model the gross geometry of manipulators.

$$A \oplus B = \{ a + b \mid a \in A, b \in B \}$$
$$\ominus A = \{ -a \mid a \in A \}$$

If a set A consists of a single point a, then $a \oplus B = \{ a \} \oplus B = A \oplus B$. Also, $A \ominus B = A \oplus (\ominus B)$. Note that, typically $A \oplus A \neq \{ 2a \mid a \in A \}$ and $A \ominus A \neq \emptyset$, although $A \oplus B = B \oplus A$. The set difference and set complement operations will be denoted $A - B$ and $-A$ respectively.

Rigid objects will be represented as *sets of possibly overlapping convex polyhedra* since this representation simplifies the algorithms for computing CO. Theorem 1 below follows directly from the definition of CO; it justifies the use of this object representation.

Theorem 1: If $A = \bigcup_{i=1}^{k_A} A_i$ and $B = \bigcup_{j=1}^{k_B} B_j$:

$$CO_A(B) = \bigcup_{i=1}^{k_A} \bigcup_{j=1}^{k_B} CO_{A_i}(B_j).$$

The position and orientation of a polyhedron will be defined relative to an initial position and orientation. In this initial position, some vertex of the polyhedron coincides with the origin of the global coordinate frame. For a polyhedron P, this vertex is called the *reference vertex of* P, or rv_P.

In the sequel, a different kind of object, called *Linked polyhedra*, is used. These objects are kinematic chains with polyhedral links and prismatic or rotary joints[4] Figure 6. The relative position

[4]Joints are represented abstractly, i.e. their representation as polyhedra do not determine their motion properties. In particular, some values of the joint parameters may cause overlap of adjacent links.

and orientation of adjacent links, A_i and A_{i+1}, is determined by the i^{th} *joint parameter (angle)* [26]. The set of joint parameters of a linked polyhedron completely specifies the position and orientation of all the links.

5. Configuration Space and *Cspace* Obstacles

The *configuration* of a k-dimensional polyhedron, A, is a point $a = (\gamma_1, \ldots, \gamma_d) \in \Re^d$, with $d = k + \binom{k}{2}$; where $(\gamma_1, \ldots, \gamma_k)$is the position of rv_A and $(\gamma_{k+1}, \ldots, \gamma_d)$ are the Euler angles specifying the orientation of A relative to its initial orientation. The configuration of a linked polyhedron having d joints is the d-vector of the joint parameters. The d-dimensional space of configurations of A is denoted $Cspace_A$. A in configuration x is $(A)_x$; A in its initial configuration is $(A)_0$.

The fundamental observation about Configuration Space is that, in $Cspace_A$, the $(A)_x$ is represented by the vector x. Given this, the basic problem in the *Cspace* approach to spatial planning is to define how the obstacles B_i map into $Cspace_A$. The mapping chosen here exploits two fundamental properties of objects: Their *rigidity*, which allows their configurations to be characterized by a few parameters, and their *solidity*, which requires that a point not be inside more than one object.

Definition: The $Cspace_A$ *obstacle* due to B, denoted $CO_A(B)$, is defined as follows:

$$CO_A(B) \equiv \{\, x \in Cspace_A \mid (A)_x \cap B \neq \emptyset \,\}$$

Thus, if $x \in CO_A(B)$ then $(A)_x$ and B either touch or overlap. Conversely, any configuration $x \notin CO_A(X)$ (for all obstacles X) is safe. The following defines a $Cspace_A$ entity complementary to $CO_A(B)$.

Definition: The $Cspace_A$ *interior* of B, denoted $CI_A(B)$, is defined as follows:

$$CI_A(B) \equiv \{\, x \in Cspace_A \mid (A)_x \subseteq B \,\}$$

The sets $CO_A(B)$ and $CI_A(B)$ are difficult to compute and manipulate since they are curved, 6-dimensional objects when A and B are polyhedra. Instead, this paper will deal with projections of slices and cross-sections[5] of $CO_A(B)$. For example, for fixed Euler angles of A, the $Cspace_A$

[5] A cross-section is a slice whose lower and upper bounds are equal.

obstacle due to B is denoted $CO_A^{xyz}(B)$ indicating that it is a set of xyz positions, rather than the full configurations. $CO_A^{xyz}(B)$ is the projection onto the x, y, z cordinates of an Euler angle cross-section[6] In general, the superscript to CO and CI will indicate the composition of their members, e.g. $CO_A^{xy}(B)$ and $CO_A^{xy\theta}(B)$ denote sets of (x, y) and (x, y, θ) values respectively.

6. The Sliding Algorithm for CO^{xy}

This section presents a "naive" algorithm for computing $CO_A^{xy}(B)$ when A and B are convex polygons and the orientation of A is fixed. In subsequent sections, more efficient algorithms are presented for this case.

When the orientation of A is fixed, the configurations of A are simply the positions of the reference vertex rv_A. Clearly, the boundary of $CO_A(B)$ is the locus of rv_A where A just touches B. This suggests a simple algorithm for computing $CO_A^{xy}(B)$: slide A around the perimeter of B and trace the path of rv_A. The term *sliding* is merely suggestive; in practice, knowing which vertex first touches an edge completely defines the path of rv_A over that edge.

The central step of this *Sliding algorithm* is to determine what point (or edge) of A first contacts each edge of B and *vice versa*, Figure 7. The normal vector of each edge of B defines an approach direction for the edge. If A is moving from a great distance towards an edge of B along the normal direction, and no edge of A is parallel to one of B, then contact will first happen at a vertex of A. This *contact vertex* is the one with the minimum perpendicular distance to the edge of B. As this vertex "slides" along the edge of B, rv_A traces an edge of $CO_A^{xy}(B)$ which is parallel to the edge of B and of equal length. But this edge is displaced by the distance from the contact vertex to rv_A, projected along the normal to the edge of B. Interchanging the roles of A and B shows that each edge of A and some vertex of B gives rise to an edge of $CO_A^{xy}(B)$. This new edge is traced out by rv_A as the edge of A "slides" along the contact vertex of B; therefore, the new edge is parallel to the edge of A and of equal length, but displaced from the contact vertex of B by the perpendicular distance between the edge and rv_A. If A and B have pairs of parallel edges, then $CO_A^{xy}(B)$ will have a parallel edge for each such pair, displaced as above, but whose length is the sum of the two edges.

[6]When A's orientation is fixed, we assume without loss of generality that A is in its initial orientation, i.e. A is simply $(A)_0$ displaced by some 3-vector x.

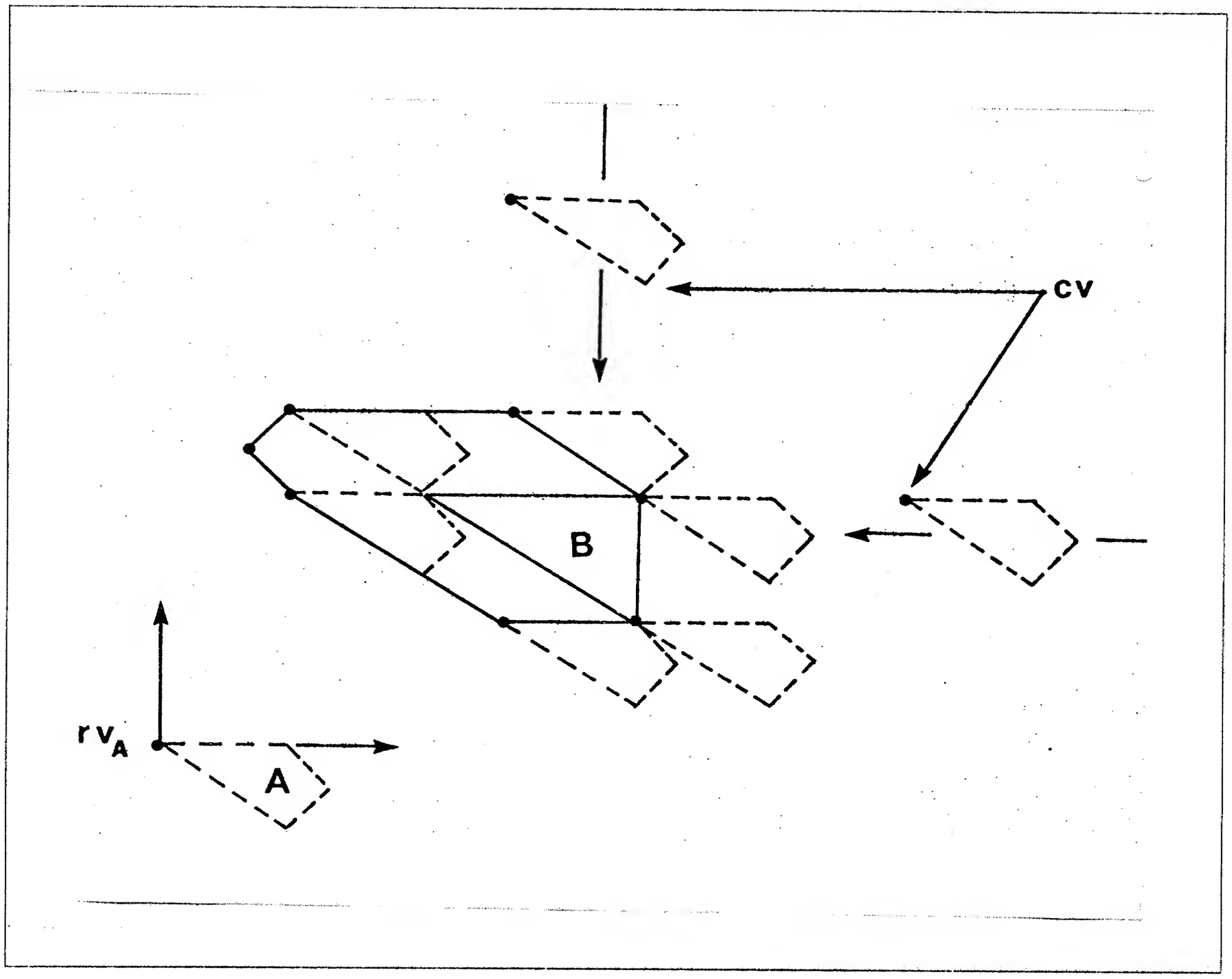


Figure 7. Illustration of Sliding algorithm, *cv* are contact vertices.

The Sliding algorithm needs $O(|edges(A)| \times |edges(B)|)$ operations to compute $CO_A^{xy}(B)$ for convex A and B. This is not optimal; Section 8 describes an $O(|edges(A)| + |edges(B)|)$ algorithm for this task.

The Sliding algorithm derives the edges of $CO_A^{xy}(B)$ from the interaction of one of (a) an edge of B and a vertex of A, (b) an edge of A and a vertex of B, or (c) an edge of A and an edge of B. Similarly, each face of $CO_A^{xyz}(B)$, for A and B convex polyhedra, can be computed from the interaction of one of

a. a face of A and a vertex of B or a face of B and a vertex of A,

b. a face of A and a coplanar edge of B or *vice versa*,

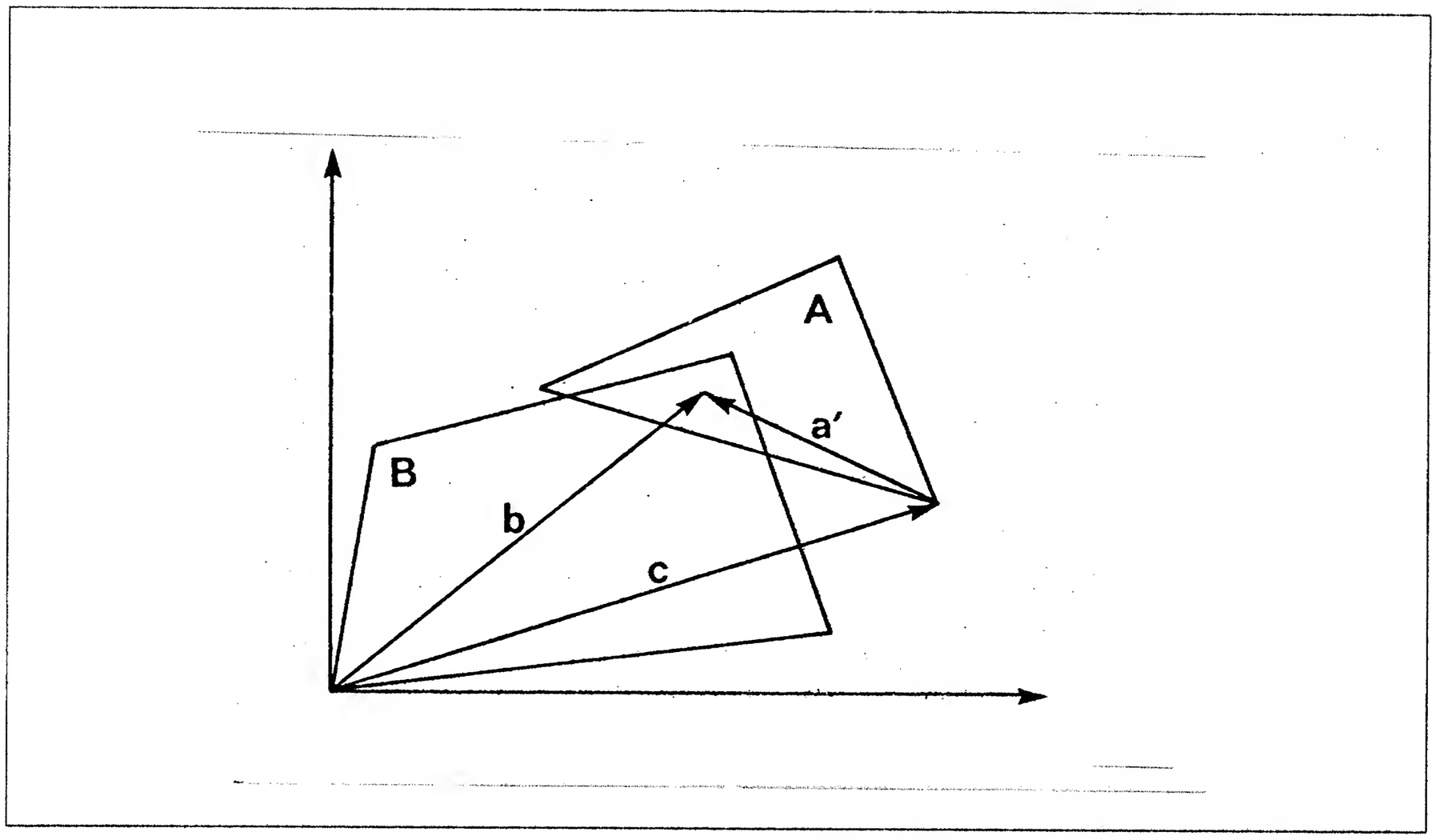


Figure 8. Illustration of the Proof of Theorem 2. Any location of rv_A, in this case c, for which A and B have a point in common (expressible as b and a'), can be expressed as $c = b - a'$. Therefore, $CO_A^{xyz}(B) = B \ominus (A)_0$.

c. an edge of A and a non-parallel edge of B,

d. a face of A and a coplanar face of B.

7. Vector Set Sums and Configuration Space Obstacles

The fundamental result of this section is the following:

Theorem 2: For A and B, sets in $\Re^3$, $CO_A^{xyz}(B) = B \ominus (A)_0$.

Proof: If c is an (x, y, z) configuration of A then $(A)_c = c \oplus (A)_0$. Therefore, if $a \in (A)_c$ then $a = a' + c$, where $a' \in (A)_0$, see Figure 8. If $b \in B \cap (A)_c$, then $b = a' + c$ and therefore $c = b - a'$. Clearly, the converse is also true.

∎

If A and B are convex then $A \oplus B$ and $A \ominus B$ are also convex [13 p.9], therefore $CO_A^{xyz}(B)$ is convex. Also, for B and A in their initial configurations,

$$CO_B^{xyz}((A)_0) = ((A)_0 \ominus (B)_0) = \ominus CO_A^{xyz}((B)_0).$$

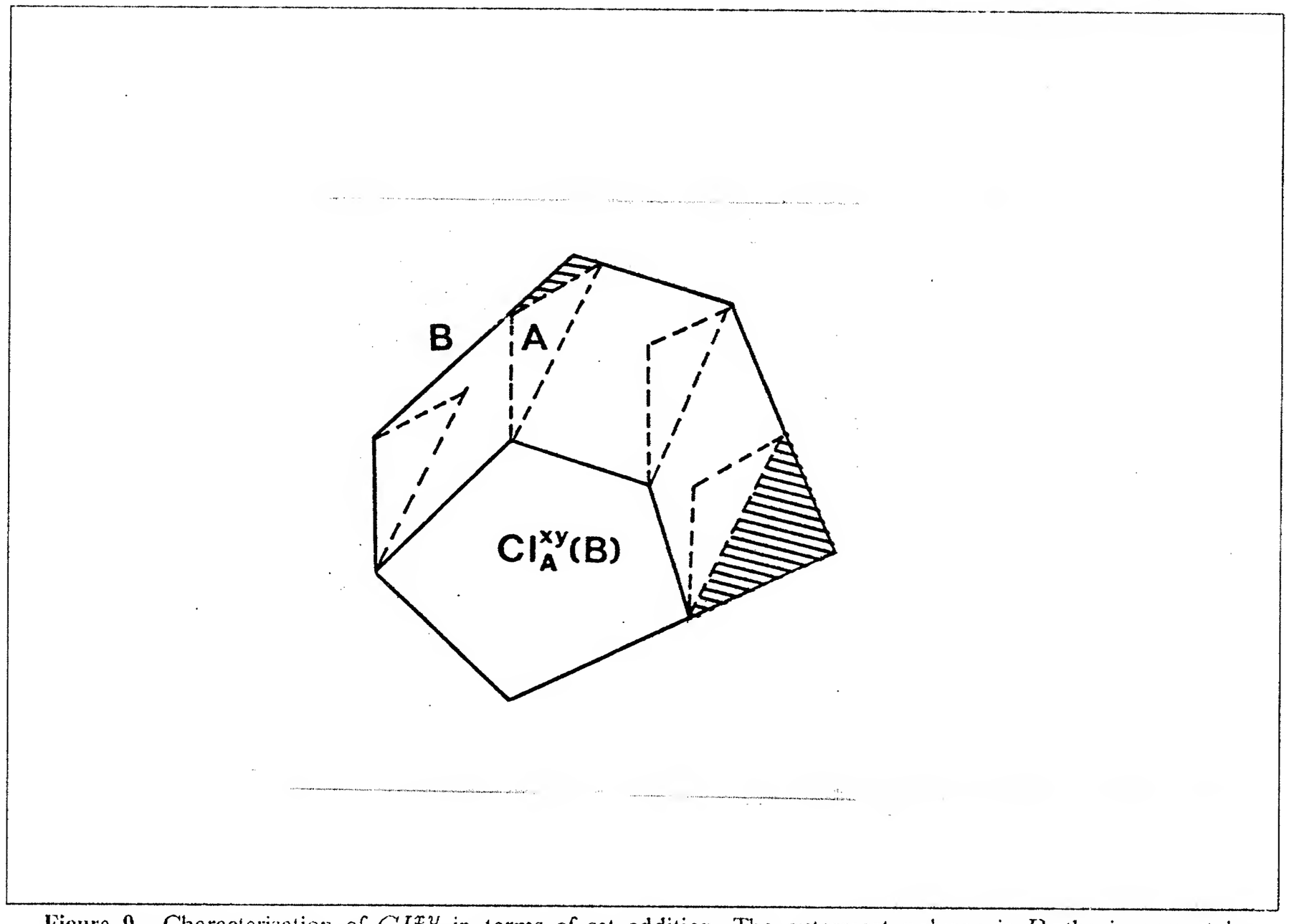


Figure 9. Characterization of CI^{xy} in terms of set addition. The outermost polygon is B, the innermost is $CI_A^{xy}(B)$. The dashed polygons are copies of $(A)_0$ placed at vertices of B, therefore the convex hull of the inner polygon and these dashed polygons is $CI_A^{xy}(B) \oplus (A)_0$ by Theorem 4. Note that $CI_A^{xy}(B) \oplus (A)_0 \neq B$, the shaded regions are $B - (CI_A^{xy}(B) \oplus (A)_0)$.

A related characterization of $CI_A^{xyz}(B)$ is also possible. For c an (x, y, z) configuration, if $c \in CI_A^{xyz}(B)$, then $c \oplus (A)_0 \subseteq B$. Since $CI_A^{xyz}(B)$ is the set of *all* such c, then $CI_A^{xyz}(B) \oplus (A)_0 \subseteq B$ and furthermore $CI_A^{xyz}(B)$ is the maximal such set, see (a) and (b) below. Clearly, if $B = X \oplus (A)_0$ then $X = CI_A^{xyz}(B)$, see (c). Figure 8 illustrates these results.

Theorem 3: For A and B convex polyhedra,

(a) $\forall X : X \oplus (A)_0 \subseteq B \Leftrightarrow X \subseteq CI_A^{xyz}(B)$

(b) $CI_A^{xyz}(B) \oplus (A)_0 \subseteq B$

(c) $CI_A^{xyz}(CO_{\ominus A}^{xyz}(B)) = B$

Note also that $CI_A(B) = -CO_A(-B)$, i.e. for A to be inside B, it must be outside of B's complement.

8. Algorithms for $CO_A^{xyz}(B)$ and $CO_A^{xy}(B)$

Theorem 4 provides a way of computing $CO_A^{xyz}(B)$ exactly for convex A and B. In addition, it provides an approximation technique for $CO_A^{xyz}(B)$ when A and B are non-convex.

Theorem 4: For polyhedra A and B,

$$conv(A \oplus B) = conv(A) \oplus conv(B) = conv\big(vert(A) \oplus vert(B)\big)$$

where $conv(X)$ denotes the convex hull [7] of set X and $vert(X)$ is the set of vertices of X.

Proof: First show that $conv(A \oplus B) = conv(A) \oplus conv(B)$.

($\supseteq$)

The definition of convex hull states that any $a \in conv(A)$ can be expressed as an affine combination of points in A. This may also be done for any $b \in conv(B)$. If $x \in conv(A) \oplus conv(B)$, $a \in conv(A), b \in conv(B), a_i \in A, b_i \in B, \sum_i \gamma_i = 1, \gamma_i \geq 0, \sum_j \beta_j = 1$ and $\beta_j \geq 0$ then

$$x = a + b = \Big(\sum_i \gamma_i a_i\Big) + b = \sum_i \gamma_i (a_i + b)$$

$$= \sum_i \gamma_i \Big(a_i + \sum_j \beta_j b_j\Big) = \sum_i \gamma_i \Big(\sum_j \beta_j a_i + \sum_j \beta_j b_j\Big)$$

$$= \sum_i \gamma_i \sum_j \beta_j (a_i + b_j) = \sum_i \sum_j \gamma_i \beta_j (a_i + b_j)$$

But, since $\sum_i \sum_j \gamma_i \beta_j = 1$ and $\gamma_i \beta_i \geq 0$, x is an affine combination of points in $A \oplus B$ and therefore belongs to its convex hull. Therefore $conv(A) \oplus conv(B) \subseteq conv(A \oplus B)$.

($\subseteq$)

If $x \in conv(A \oplus B)$, then for $a_i \in A$ and $b_i \in B$,

$$x = \sum_i \gamma_i (a_i + b_i) = \sum_i \gamma_i a_i + \sum_i \gamma_i b_i.$$

Therefore, $x \in conv(A) \oplus conv(B)$.

This establishes that $conv(A \oplus B) = conv(A) \oplus conv(B)$. Replacing $vert(A)$ for A and $vert(B)$ for B and using the fact that $conv(A) = conv(vert(A))$ [13], shows that $conv(vert(A) \oplus vert(B)) = conv(A) \oplus conv(B)$.

∎

[7] The *convex hull*, $conv(A)$, of a nonempty set $A \subseteq \Re^d$ is $\{\sum_{i=1}^n \gamma_i x_i \mid x_i \in A, \gamma_i \geq 0, \sum_{i=1}^n \gamma_i = 1, n = 1, 2, \ldots\}$ [13 p.15]

Corollary: For convex polyhedra A and B, $A \oplus B = conv\big(vert(A) \oplus vert(B)\big)$ and therefore $CO_A^{xyz}(B) = conv\big(vert(B) \ominus vert((A)_0)\big)$.

Proof: The first part of the corollary follows directly from the fact that, for convex A, $A = conv(A)$. The second part follows from Theorem 2.

∎

Many algorithms exist for finding the convex hull of a finite set of points on the plane, e.g. [9] [12] [15] [29]. [29] also describe an efficient algorithm for points in $\Re^3$. These algorithms are known to run in worst-case time $O(v \log v)$, where v is the size of the input set. Therefore, Theorem 4 leads immediately to an algorithm for CO^{xyz} and an upper bound on the computational complexity of the problem.

Theorem 5: For convex $A, B \subseteq \Re^3$, $CO_A^{xyz}(B)$ can be computed in time $O(n^2 \log n)$; where $n = |vert(A)| + |vert(B)|$.

Proof: The set $vert(B) \ominus vert((A)_0)$ is of size $|vert(A)| \times |vert(B)|$, i.e. $O(n^2)$. Applying an $O(v \log v)$ convex hull algorithm to this set gives an $O(n^2 \log n)$ algorithm for computing $CO_A^{xyz}(B)$. This result holds only for convex polyhedra of dimension $k \leq 3$ [7].

∎

The algorithm of Theorem 5 is not optimal; an $O(n)$ algorithm exists for $CO_A^{xy}(B)$ when A and B are convex polygons[8] :

Definition: $\pi(A, u)$ denotes the *supporting plane (line)* of A with outward normal u. $\pi(A, u)$ contains at least one boundary point of A, call it a, and for any $a' \in A$ then $\langle a', u\rangle \leq \langle a, u\rangle$. Thus, all of A is in one of the closed half spaces bounded by $\pi(A, u)$ and u points away from the interior of A.

Lemma 1: If A and B are convex sets then

$$\pi(A \oplus B, u) \cap (A \oplus B) = \big(\pi(A, u) \cap A\big) \oplus \big(\pi(B, u) \cap B\big) \tag{1}$$

[8]The development in this section is based on that in Section 14 of [4]

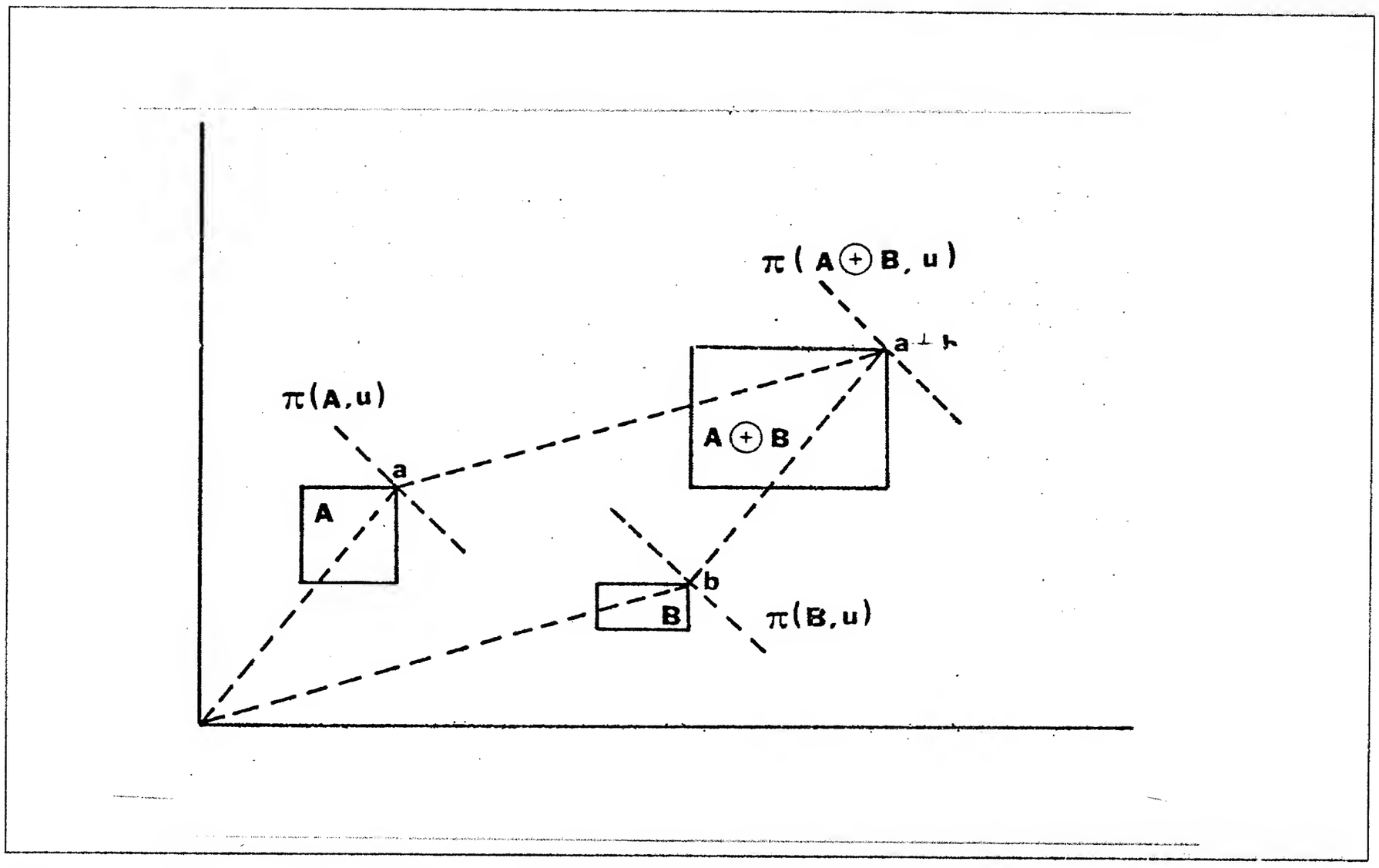


Figure 10. Illustration for Lemma 1.

Lemma 2:

(a) Let $s(a_1, a_2)$ be a line segment and b a point, then $s(a_1, a_2) \oplus b = s(a_1 + b, a_2 + b)$ is a line segment parallel to $s(a_1, a_2)$ and of equal length. See Figure 11(a).

(b) Let $s(a_1, a_2)$ and $s(b_1, b_2)$ be parallel line segments such that $(a_2 - a_1) = k(b_2 - b_1)$ for $k > 0$. Then $s(a_1, a_2) \oplus s(b_1, b_2) = s(a_1 + b_1, a_2 + b_2)$ and the length of the sum is the sum of the lengths of the summands. See Figure 11(b).

Theorem 6: For convex polygons A and B, $CO_A^{xy}(B)$ can be computed in worst case time $O(|vert(A)| + |vert(B)|)$.

Proof: For fixed u, each term on the right hand side of (1) is either a line segment (edge) or a single point (vertex), it follows from Lemma 2 that the term on the left is one of:

a. a new vertex, when two vertices are combined;

b. a displaced edge, when an edge and a vertex are combined (Lemma 2a);

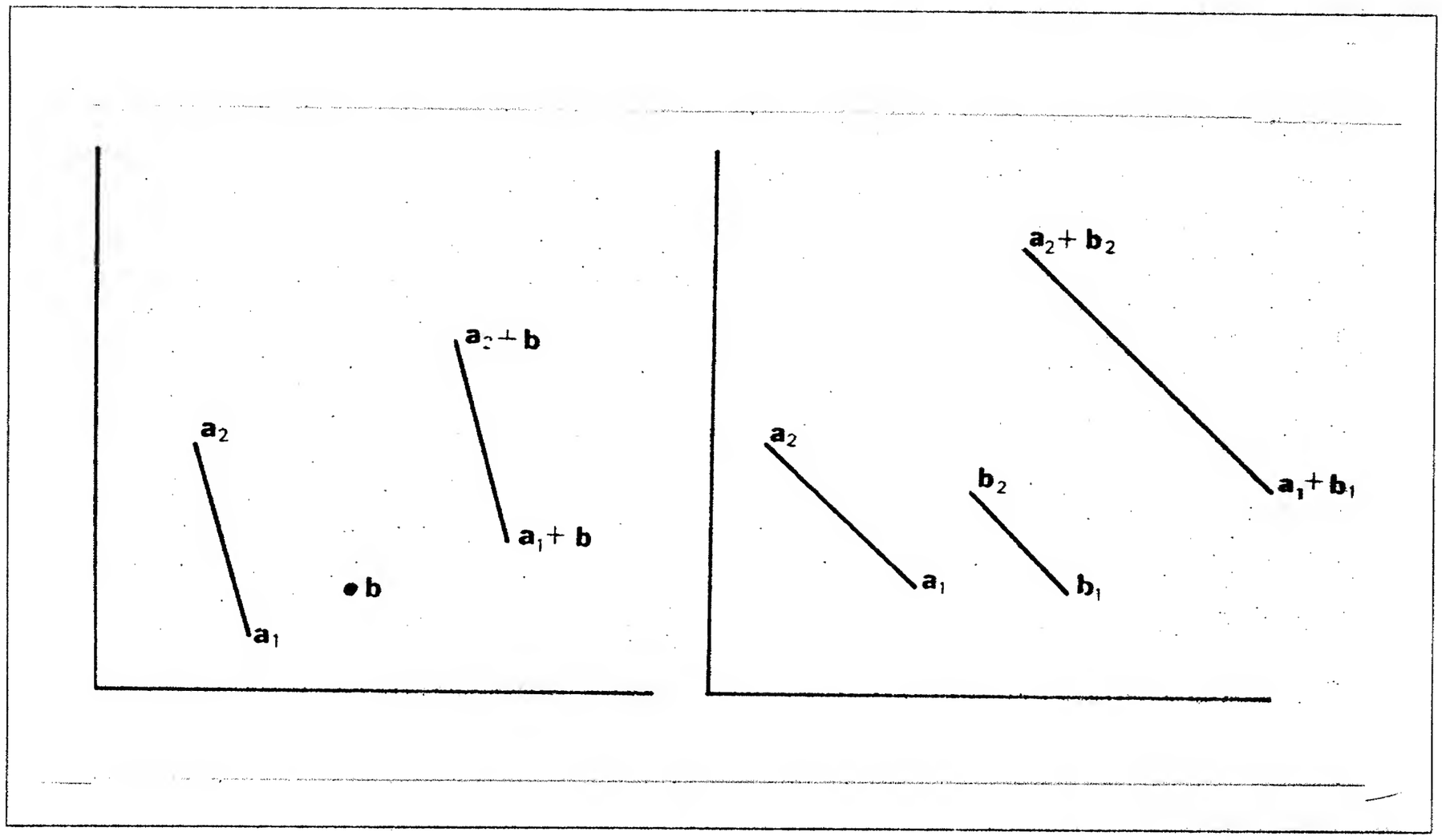


Figure 11. Illustration for Lemma 2.

c. a pair of displaced end-to-end edges, when two edges are combined (Lemma 2b).

As u rotates counterclockwise, the boundary of $A \oplus B$ is formed by joining a succession of these line segments. Note that, because of the convexity of A and B, each edge is encountered exactly once [22 p.13].

A polygon is stored as a list of vertices in the same order as they are encountered by the counterclockwise sweep of u. This is equivalent to a total order on the edges, based on the angle that the edge makes with the x axis. For a polygon P, assume the j^{th} edge in this order, $e_j == s(v_j, v_{j+1})$, makes the angle θ_j, then

$$\pi(P,u) \cap P = \begin{cases} v_j, & \text{if } \theta_{j-1} < \theta(u) < \theta_j \\ e_j, & \text{if } \theta(u) = \theta_j \\ v_{j+1}, & \text{if } \theta_j < \theta(u) < \theta_{j+1} \end{cases}$$

The time for constructing the new vertices is bounded by a constant, since it involves at most two vector additions. Thus $A \oplus B$ can be computed in linear time during a scan of the vertices of A and

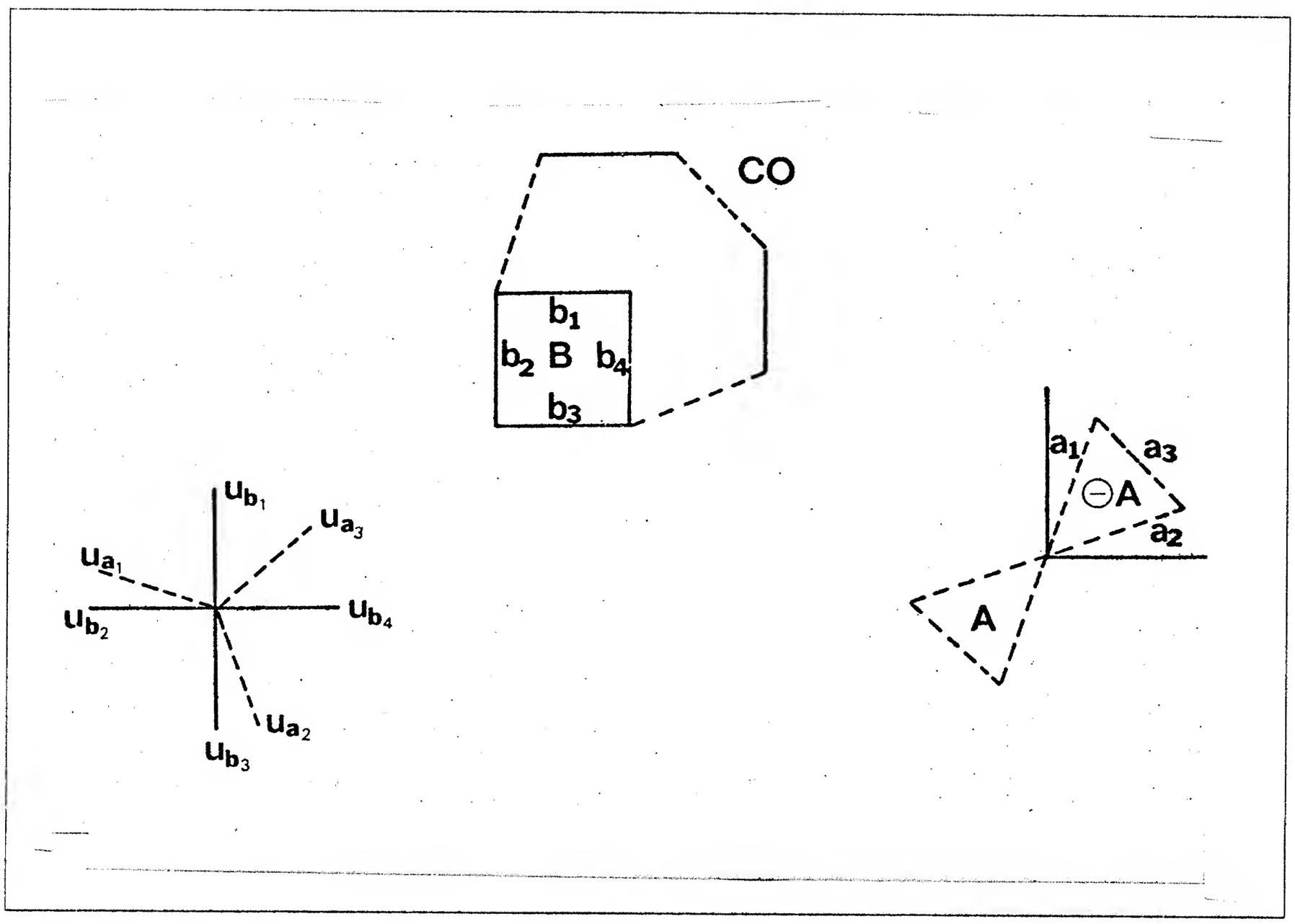


Figure 12. The edges of $B \ominus (A)_0$, when A and B are convex polygons, are found by merging the edge lists of B and $\ominus(A)_0$, ordered on the angle their normals make with the positive x axis.

B, Figure 12. An implementation of this operation is shown in Appendix 1. Similarly $B \ominus (A)_0$ can be computed in linear time by first converting each vertex a_i to $rv_A - a_i$, Figure 12.

When A and/or B are non-convex polygons, $CO_A^{xy}(B)$ can be computed by an extension of the algorithm above. The method relies on decomposing the boundaries of the polygons into a sequence of polygonal arcs whose internal angles, i.e. the angle facing the inside of the polygon, are each less than π. The algorithm of Theorem 6 can then be applied to pairs of arcs; the result is a polygon whose boundary, in general, intersects itself. The algorithm requires, in the worst case, $O(|edges(A)| \times |edges(B)|)$ steps [20].

9. Dealing with Rotations of A

If A and B are polygons, then $CO_A(B)$ is an object in $\Re^3$, denoted $CO_A^{xy\theta}(B)$. The shape of $CO_A^{xy\theta}(B)$, when A and B are convex, will be investigated below by examining changes in the cross-sections[9] of $CO_A^{xy\theta}$ as θ changes.

Assume a fixed value for θ. If A and B have no parallel edges, the proof of Theorem 6 shows that each edge of $CO_A^{xy}(B)$ can be expressed as one of:

$$e_i^a = b_j \oplus s\big(a_i(\theta), a_{i+1}(\theta)\big) \tag{2a}$$

$$e_j^b = a_i(\theta) \oplus s(b_j, b_{j+1}) \tag{2b}$$

In these expressions, b_j is the position vector of the j^{th} member of $vert(B)$ and $a_i(\theta)$ is the position of the i^{th} member of $vert(\ominus(A)_0)$. which depends on θ. The order in which the a_i and b_j are encountered in the counterclockwise scan described in Theorem 6 determines the (i, j) pairings of vertices and edges.

Equation (2) shows that, for small changes in θ, the e_i^a rotate around b_j, while the e_j^b are simply displaced, Figure 13. In addition to these changes in the xy-cross-section of $CO_A^{xy\theta}$, there are discontinuous changes at values of θ, denoted θ_i^*, where an edge from A becomes parallel to one from B. For values of θ just greater than these θ_i^*, this pair of edges has a different order in the scan of Theorem 6 from what they had when θ was just less than θ_i^*. Therefore, the (i, j) pairings between edges and vertices changes. There are $O(|edges(A)| \times |edges(B)|)$ such θ_i^* in $CO_A^{xy\theta}$.

Between discontinuities, the lines defined by e_j^b edges have a simple dependence on θ. The edge $s(b_j, b_{j+1})$ is on a line whose vector equation is: $\langle x, u_j\rangle = \langle b_j, u_j\rangle$ where u_j is the constant unit normal to $s(b_j, b_{j+1})$. Let $a_i(\theta)$ make the angle $\theta + \eta_i$ with the x axis, with η_i constant, and u_j make the angle ϕ_j with the x axis. Then, the equation for the line including e_j^b is

$$\begin{aligned}\langle u_j, x\rangle &= \langle u_j, a_i(\theta) + b_j\rangle \\ &= \|a_i\| \cos(\theta + \eta_i - \phi_j) + \langle b_j, u_j\rangle\end{aligned} \tag{3}$$

The terms are illustrated in Figure 14. This equation holds only for $\theta \in [\phi_j - \eta_i, \phi_{j+1} - \eta_i]$ between discontinuities; but within that interval it defines a plane in the space $(x, y, \cos\theta)$. This space is analogous to that defined by semi-log graph paper. As long as the θ-interval is known and $\cos^{-1}$ is

[9] The cross-section of $CO_A^{xy\theta}(B)$, for constant θ, is $CO_A^{xy}(B)$.

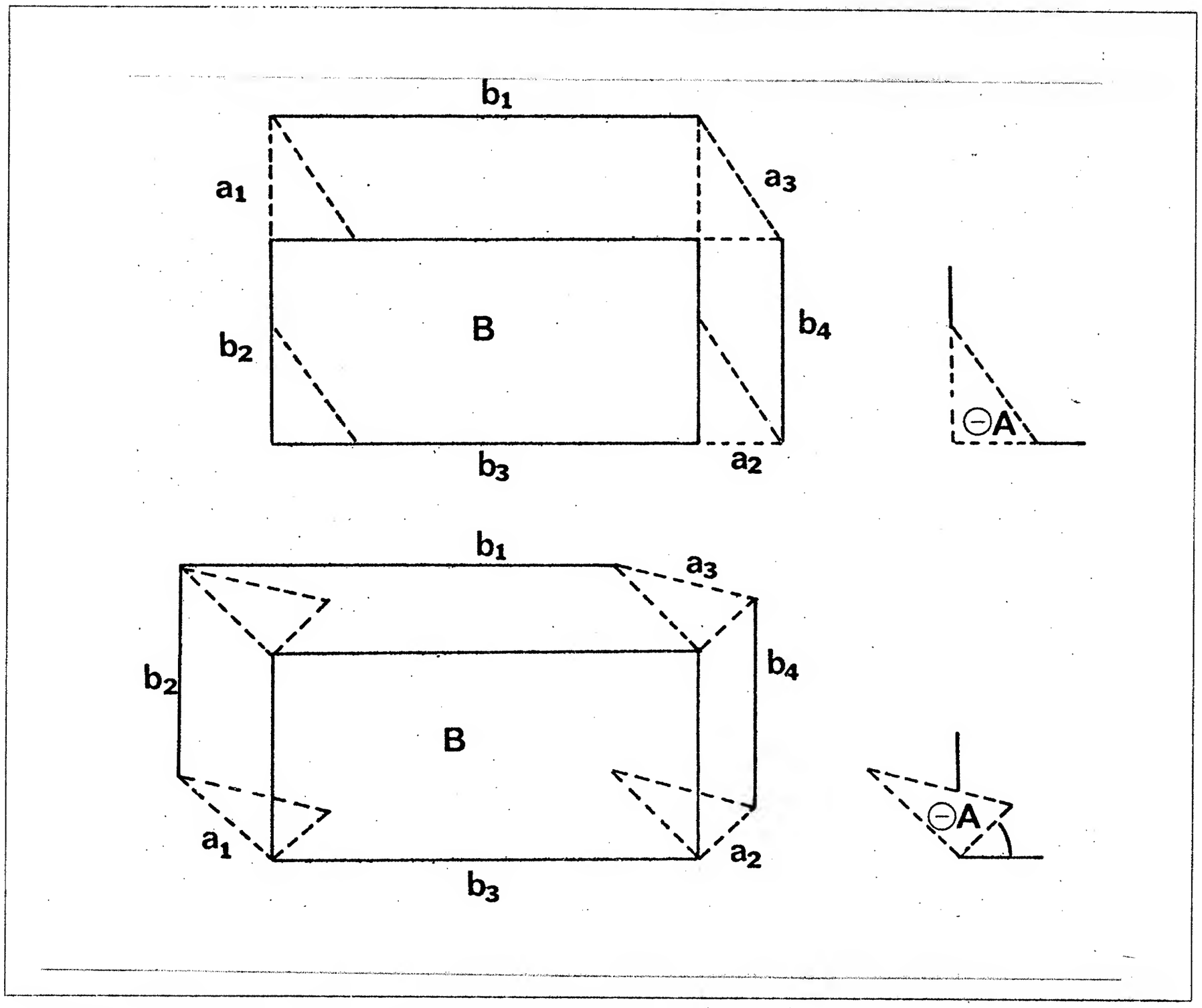


Figure 13. $CO_A^{xy}(B)$ is the convex hull of the union of $vert\left(\ominus(A)_{b_i}\right)$ for each $b_i \in vert(B)$. When A (and $\ominus(A)_0$) rotates by θ, the e_i^a rotate around b_j and the e_j^b are displaced. When an e_i^a is aligned with an e_j^b for some θ, any extra rotation will interchange the order in which they are encountered during the counterclockwise scan of Theorem 6.

single valued over the interval then the mapping from $(x, y, \cos\theta)$ to the (x, y, θ) space is unique. The θ-ranges can always be chosen so that this is the case.

The e_i^a cannot be treated in a similar fashion because the *orientation* of the edge changes with θ, i.e. the equation of the surface involve products of the form $x\cos\theta$ and $y\cos\theta$. Instead of trying to represent them exactly by non-planar surfaces, this section develops a simple approximation technique that avoids dealing directly with these edges. The technique will be illustrated first for CO^{xy} and later for $CO^{xy\theta}$.

For fixed θ, if the lines defined by the e_j^b are extended until they intersect, the resulting figure

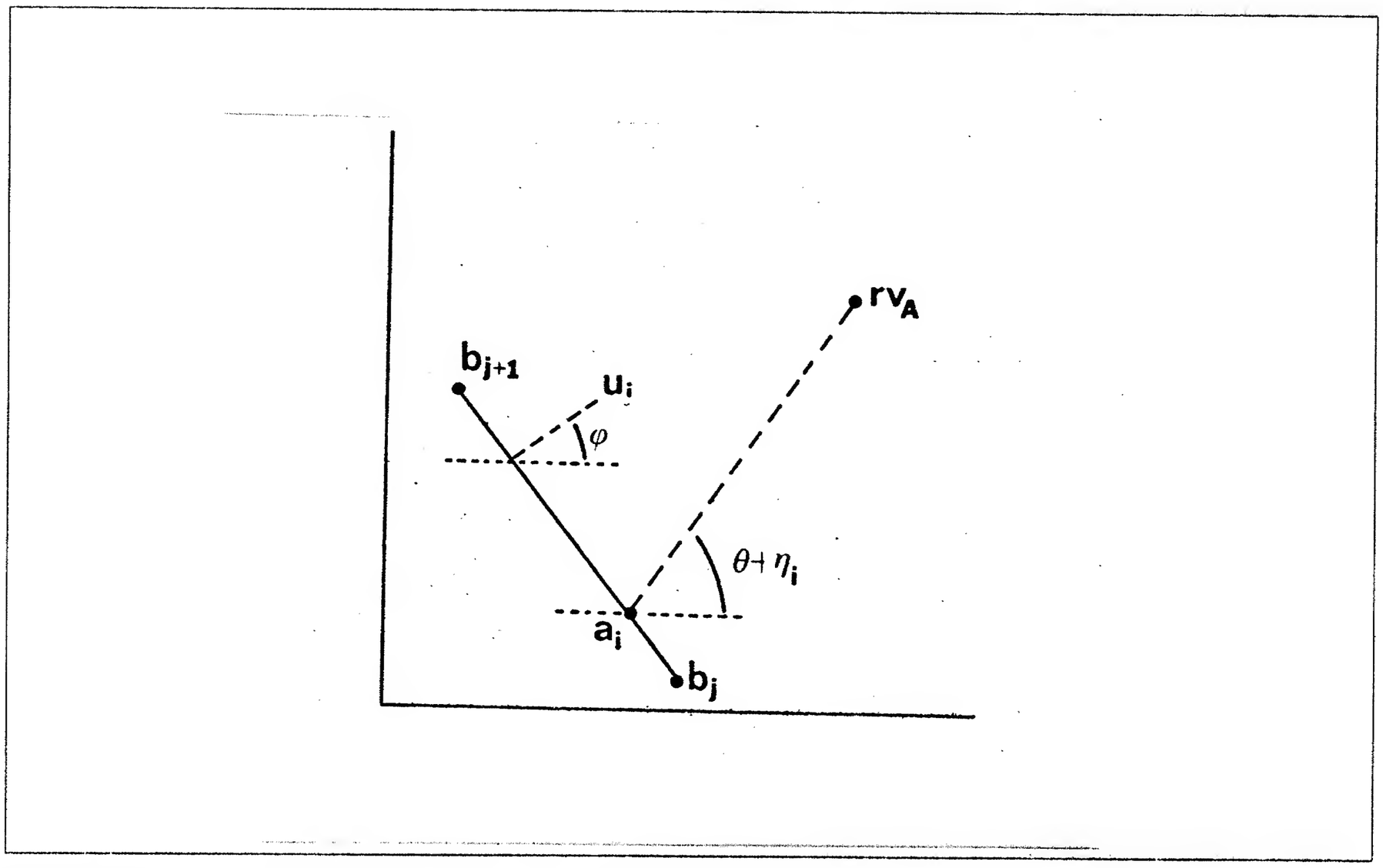


Figure 14. Illustration of terms in equation (3).

completely includes $CO_A^{xy}(B)$. This approximation can be very poor when the angle defined by adjacent e_j^b is very acute, see Figure 15. The approximation can be improved by introducing additional lines whose normals point between those of e_{j-1}^b and e_j^b. These lines should be farther from b_j than any of the e_i^a paired with b_j in **(2a)**.

This method can also be used to approximate the slice of $CO_A^{xy\theta}(B)$ between discontinuities. The e_j^b and the lines to bound the e_i^a edges both define planes in $(x, y, \cos\theta)$ space, see (3), for some range of θ-values. The boundaries of the θ-intervals also define planes whose equations are of the form $\theta = \phi_j - \eta_i$. These planes bound half-spaces whose intersection defines a convex polyhedron in $(x, y, \cos\theta)$. This polyhedron contains all of $CO_A^{xy\theta}$ within the θ-interval. Therefore, it can be used to approximate $CO_A^{xy\theta}$ over that interval. The union of the resulting convex polyhedra for each θ-interval is an approximation to $CO_A^{xy\theta}$.

The discussion above shows how to build a set of $n = |edges(B)| + |vert(B)|$ half spaces bounding $CO_A^{xy\theta}(B)$ within each θ-range. These half spaces can be intersected to construct a convex

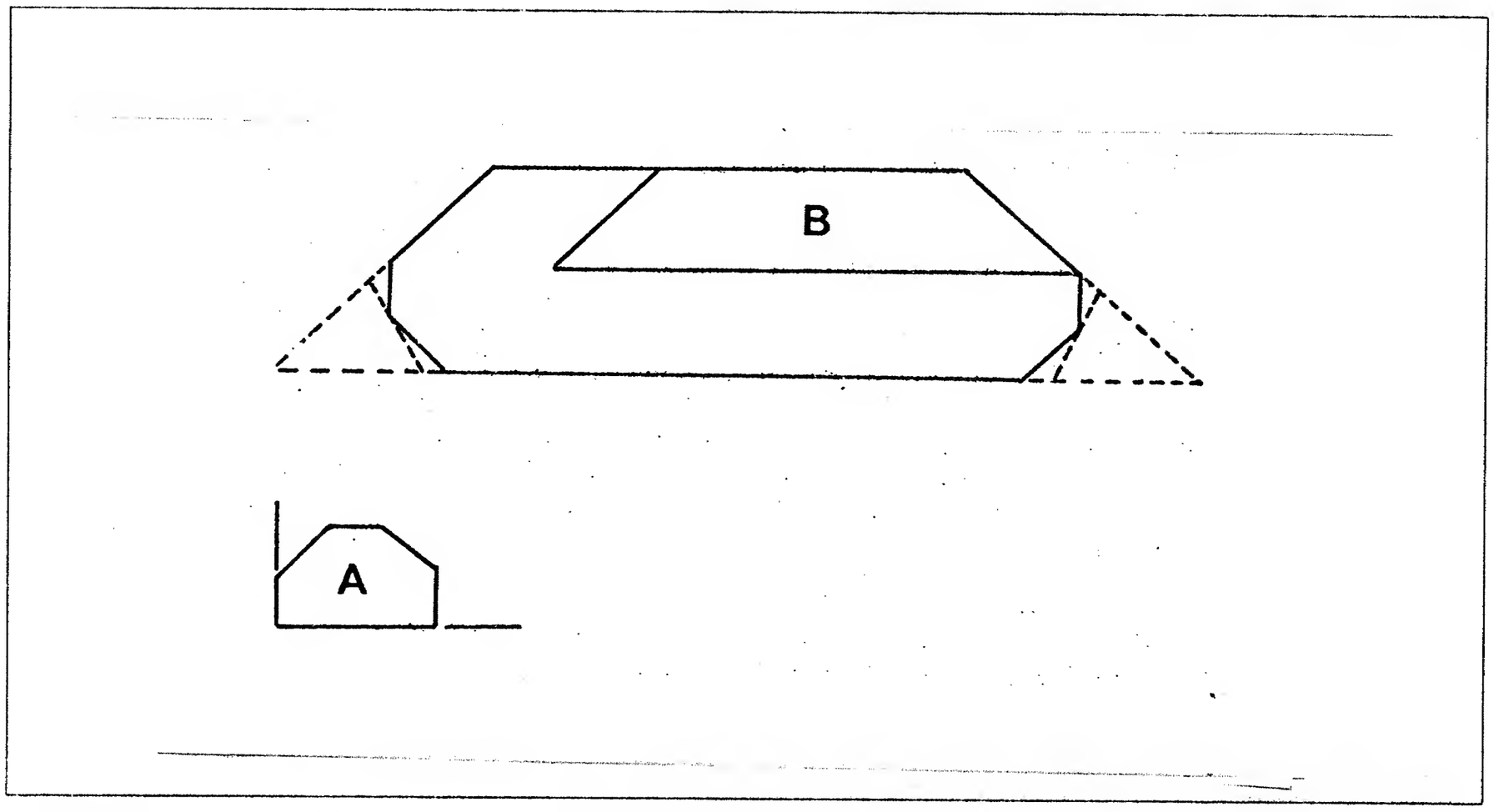


Figure 15. Approximating $CO_A^{xy}(B)$ using only the e_j^b works well at vertices with large interior angle, but poorly for small interior angles. The lines shown dashed can be used to improve the approximation.

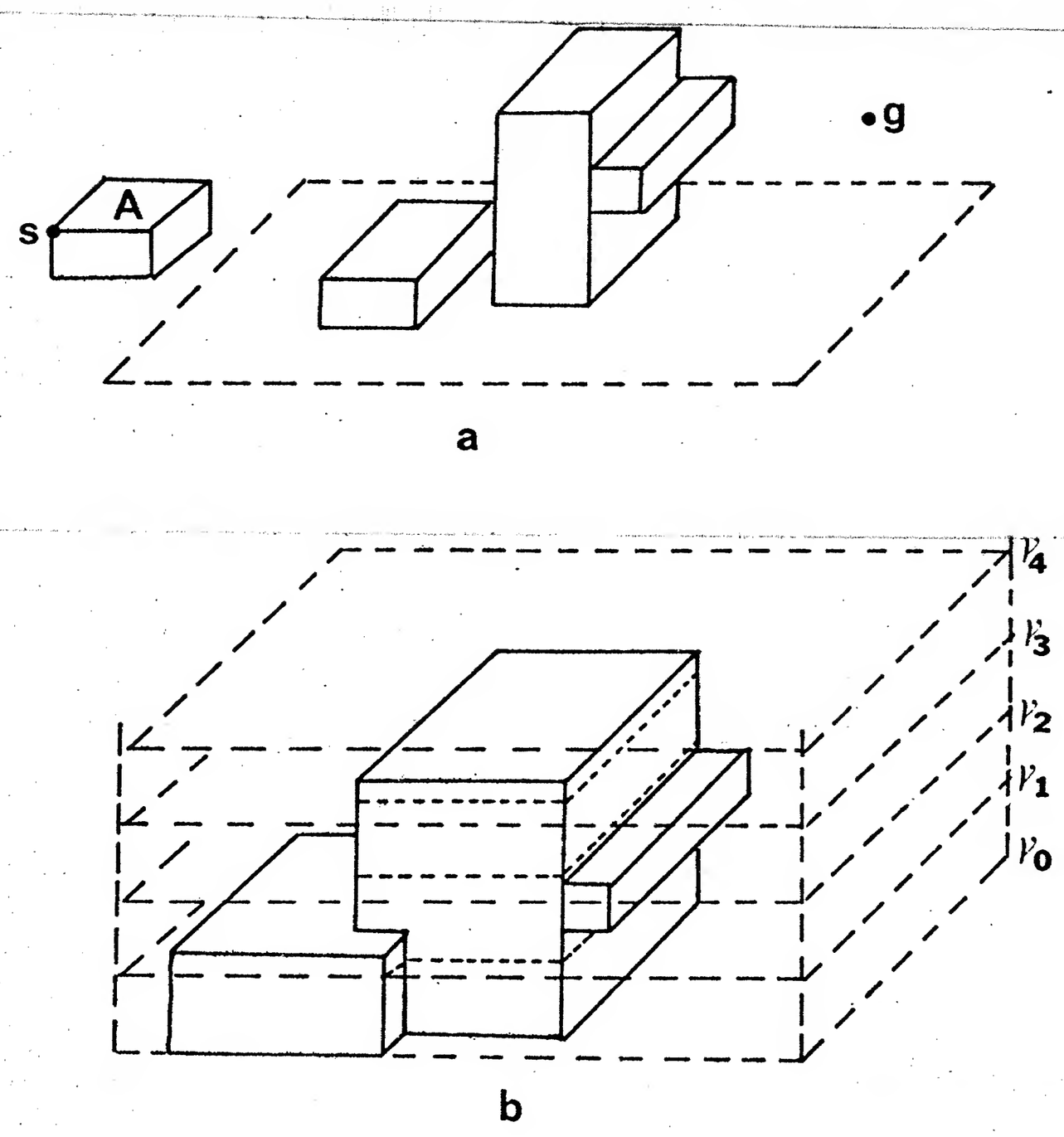
polyhedron in $O(n \log n)$ time [7] [30]. There are $O\big(|edges(A)| \times |edges(B)|\big)$ θ-ranges that need to be considered; therefore, the complete approximation may be found in $O(n^3 \log n)$ time, although in many applications, a complete approximation might not be necessary.

The same techniques can also be applied to computing $CI_A^{xy\theta}(B)$ and since the CI only has edges of the form $(2b)$, the resulting polyhedra in $(x, y, \cos\theta)$ are an exact representation of the $Cspace_A$ entity [20].

This approximation is, in principle, applicable to polyhedra in 3-dimensions; the results would be a set of polyhedra in 6-dimensions. The extension is conceptually simple, but the difficulties of deriving and representing the three-dimensional orientation constraints make the approach unattractive. The next section examines an alternative to dealing with the high-dimensional polyhedra required by this technique.

10. Approximating High Dimension *Cspace* Obstacles

Section 3 introduced the use of projections of obstacle *slices* as a means of approximating high dimensional *Cspace* Obstacles. Figure 16 shows a decomposition of $Cspace_A \subseteq \Re^3$ into a family



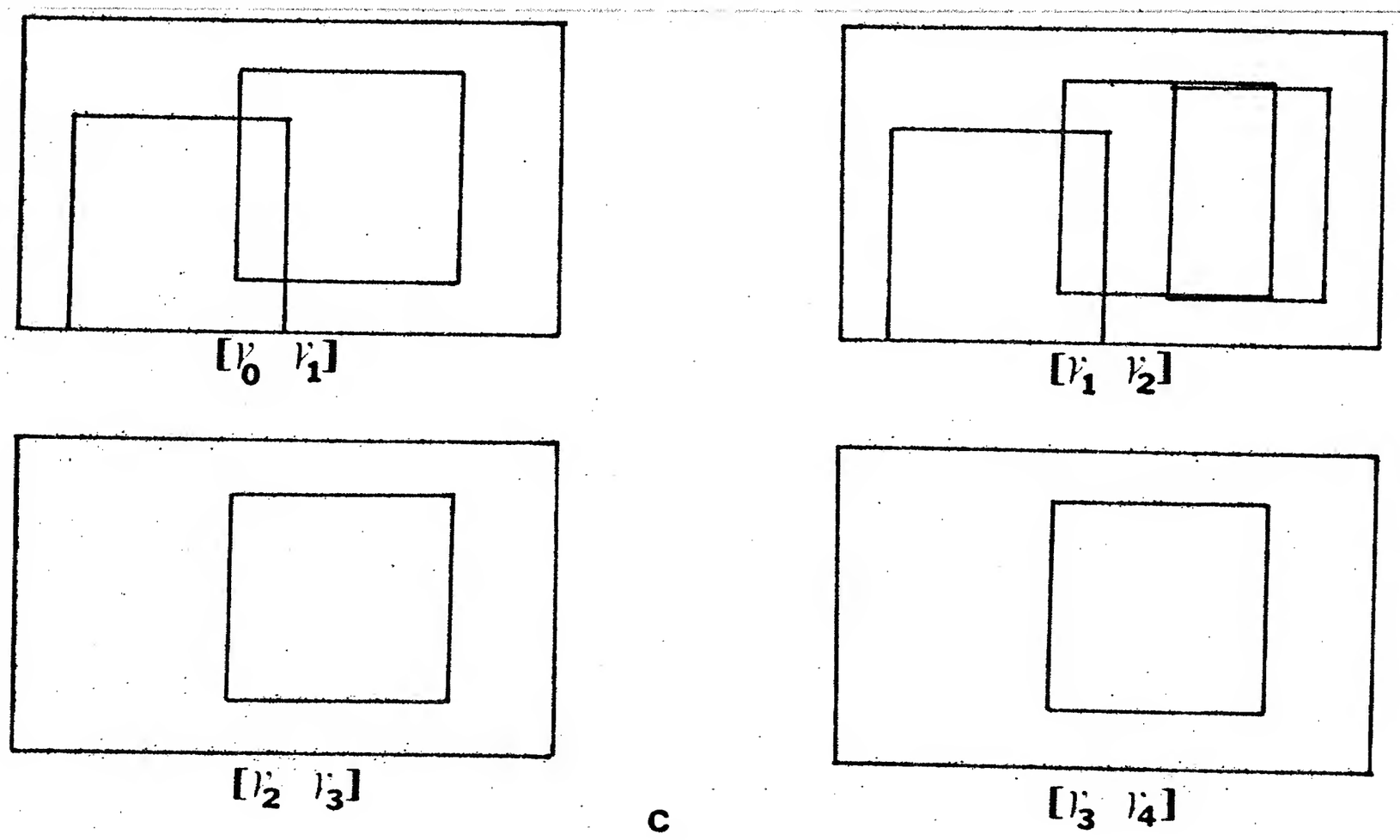


Figure 16. z slice projections. This example shows the slice projections decomposing $Cspace_A$ into four $Cspace_A[\gamma_j, \gamma_{j+1}]$.

of slices $Cspace_A[\gamma_j, \gamma_{j+1}]$. Each of these new *Cspace*s is a z slice projection of $Cspace_A$. This new family of spaces is a conservative representation of the obstacles, i.e. it represents the worst case constraints on configurations of A whose z coordinates are within the range $[\gamma_j, \gamma_{j+1}]$.

The power of the slice projection approach is that the family of slices captures all the constraints needed to plan safe paths in $Cspace_A$. Having all the slices of the $CO_A(B_i)$, there is no need to refer to the $CO_A(B_i)$ themselves. On the other hand, the algorithms in the Sections 6 through 9 are for cross-sections of $CO_A(B)$, e.g. CO_A^{xy} and $CO_A^{xy\theta}$, not for slice projection. The basic result of this section is that these algorithms can be used to compute polyhedral approximations to the slice projections.

The construction that relates slice projections to cross-section projections is the *swept volume* of an object. Intuitively, the swept volume of A is all the space that A covers when moving within a range of configurations. In particular, given two configurations for A, called c and c', then the union of $(A)_a$ for all $c \leq a \leq c'$ is the swept volume of A over the configuration range $[c, c']$. Generally, c and c' differ only on some subset, K, of the configuration coordinates. For example, if c and c' are of the form $(\beta_1, \beta_2, \beta_3)$ and $K = \{3\}$, then the swept volume of A over the range $[c, c']_K$ refers to the union of A over a set of configurations differing only on β_3. The swept volume of A over a configuration range is denoted $A[c, c']_K$.

The swept volume of A, a rigid object, is also a rigid object[10] with the same number of degrees of freedom. For linked polyhedra, the situation is not so simple, because of the interdependence of the *Cspace* parameters.

Note that for a linked polyhedron, the position of link j typically depends on the positions of links $k < j$, which are closer to the base than link j. Let $K = \{j\}$, $c = (\theta_i)$, $c' = (\theta_i')$, and $[c, c']_K$ define a range of configurations differing on the j^{th} $Cspace_A$ parameter. Since joint j varies over a range of values, links $l \geq j$ will move over a range of positions which depend on the values of c and c', as shown in Figure 17. The union of each of the link volumes over its specified range of positions is the swept volume of the linked polyhedron. The swept volume of links j through n can be taken as defining a new j^{th} link. The first $j - 1$ links and the new j^{th} link define a new manipulator whose configuration can be described by the first $j - 1$ joint parameters. On the other hand, the shape of the

[10]Note that, in general, the swept volume of a polyhedron is *not* a polyhedron, although the development relies on computing polyhedral approximations to it.

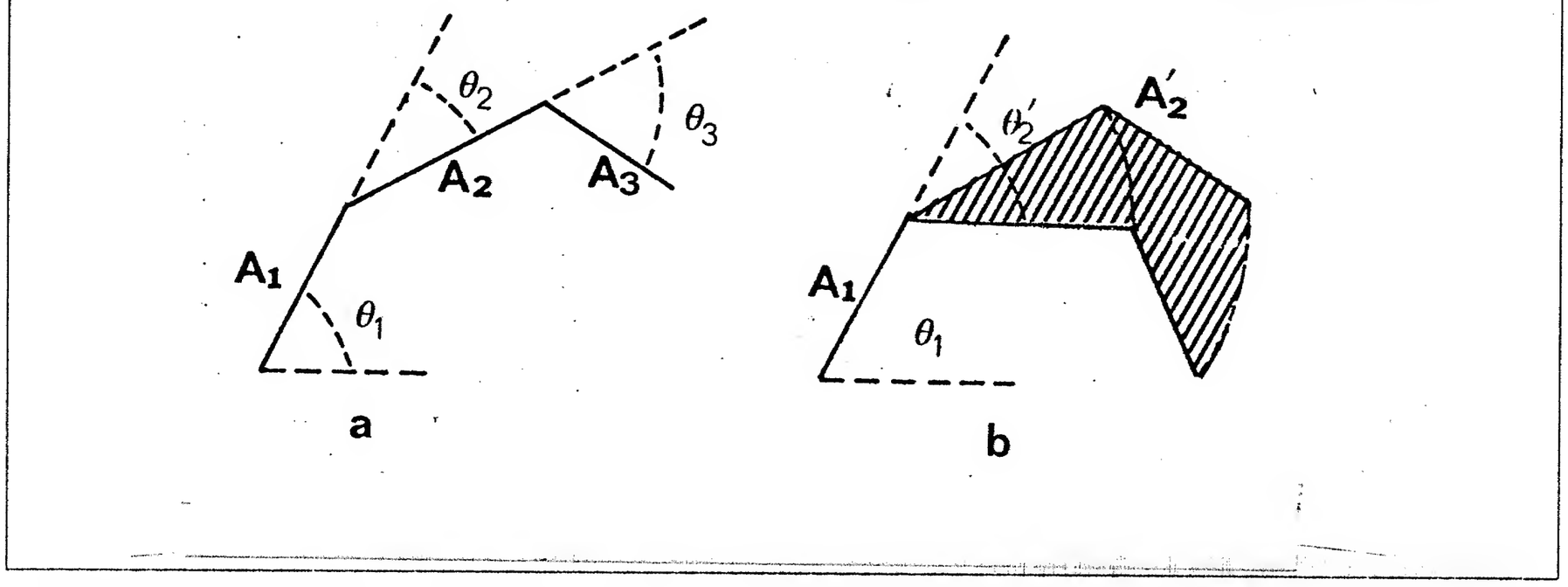


Figure 17. Changes in the second joint angle from θ_2 to θ_2' causes changes in the configurations of both link A_2 and link A_3.

new link j depends not only on the K-parameters of c and c', i.e. θ_j and θ'_j, but also on θ_l for $l > j$. This implicit dependence on parameters of c and c' that are not in K is undesirable, since it means that the *shape* of the new j^{th} link will vary. Letting $K = \{j, \ldots, n\}$, then the shape of the swept volume depends only on the K-parameters of c and c', while its configuration is determined by the $(I - K)$-parameters. A swept volume that satisfies this property is called *displaceable.*

The fact that the swept volume of a linked polyhedron A does not have the same degrees of freedom as does A forms the basis for the relationship between slice projection and cross-section projection. If the swept volume is displaceable, the $I - K$ parameters may be changed, but changes to the K parameters are not legal. Therefore, the *Cspace* of the swept volume of A is of lower dimension than the *Cspace* of A. In particular, configurations in $Cspace_A$ that have equal $I - K$ parameters and whose K parameters are in the defining range of the swept volume, project into the same configuration in $Cspace_{A[c,c']_K}$.

If $A[c, c']_K$ overlaps some obstacle B then, for some configuration a in that range, $(A)_a$ overlaps B. The converse is also true. If $A[c, c']_K$ is displaceable, then $CO_{A[c,c']_K}(B)$ is the set of $I - K$ projections of those configurations of A within $[c, c']_K$ for which A overlaps B. Equivalently, $CO_{A[c,c']_K}(B)$ is the $I - K$ projection of the $[c, c']_K$ slice of $CO_A(B)$. If the configurations of the swept volume are one of (x, y), (x, y, z), or (x, y, θ) then the algorithms of the previous sections can be used to compute

$CO_{A[c,c']_K}(B)$ and thereby compute the required slice projections[11] .

A formal statement and proof of this result is included in Appendix 2 as Theorem 8. This theorem is of practical importance since it provides the mechanism underlying the Findspace and Findpath implementations described in [20] and [21]. In addition, the proof of the theorem demonstrates the usefulness of the *Cspace* concept as a tool in theoretical analyses of spatial planning problems.

11. Related Work in Spatial Planning

The definition of the Findspace problem used here is based on that in [46]. Approaches to this problem are described by [37] and [27]. The latter, which is the more relevant, is an application of the Warnock algorithm for hidden line elimination. It involves recursively subdividing the workspace until an area "large enough" for the object is found. This approach has several drawbacks: (1) any non-overlapping subdivision strategy will break up potentially useful areas, and (2) the implementation of the predicate "large enough" is not specified.

The *Cspace* approach to Findspace and Findpath described here is an extension of that reported in [21]. In that paper, an approximate algorithm for $CO_A^{xyz}(B)$ is described and the Vgraph algorithm for high-dimensional Findpath is first presented.

The basic idea of representing position constraints as geometric figures, e.g. $CO_A^{xy}(B)$, has been used (independently) in [1], [2] and [3], who employed an algorithm to compute CO^{xy} for non-convex polygons in a technique for two dimensional layout. The template packing approach described in [10] uses a related computation based on a chain-code description of figure boundaries. [36] reports algorithms for packing of parallelopipeds in the presence of obstacles using a construct equivalent to the CO^{xy}, but defined as "the hodograph of the Close Positioning Function". The only use of this construct in the paper is for computing CO^{xy} for aligned rectangular prisms.

The work by Udupa, reported in [40] [41], was the first to approach Findpath by explicitly using transformed obstacles and a space where the moving object is a point. Udupa used only rough approximations to the actual *Cspace* obstacles and had no direct method for representing constraints on

[11] Of course, this requires computing a convex polyhedral approximation to the swept volume of A. Simple approximations are not difficult to compute [20], but this is an area where better algorithms are required. Nevertheless, the swept volume computation is a 3-dimensional operation which can be defined and executed without recourse to 6-dimensional constructs.

more than three degrees of freedom. [41] also surveys previous heuristic approaches to the Findpath problem for manipulators [17] [28] [44]. An early paper on Shakey [25] describes a technique for Findpath using a simple object transformation that defines safe points for a circular approximation to the mobile robot and uses a graph search formulation of the problem. More recent papers on navigation of mobile robots are also relevant to 2-dimensional Findpath [11] [23] [39]. [14] reports on a program for planning the path of a 2-dimensional sofa through a corridor. This program does a brute-force graph search through a quantized *Cspace*.

[31] proposes an extension of the approach in [21] to the general Findpath problem, but using an exact representation of the high-dimensional *Cspace* obstacles. The basic approach is to define the general configuration constraints as a set of multinomials in the position parameters of A. But, the proposal still requires elaboration. It defines the configuration space constraints in terms of the relationships of vertices of one object to the faces of the other. This is adequate for polygons, but the equations in the paper only express the constraints necessary for vertices of A to be outside of B, i.e. they are of the form of (3) above. They do not account for the positions of A where vertices of B are in contact with A. Thus, the equations do not represent the correct constraints on the position of A. The new equations will have terms of the form $x\cos\theta$ and $y\cos\theta$. Furthermore, the approach of defining the configuration constraints by examining the interaction of vertices and faces does not generalize to 3-dimensional polyhedra. It is not enough to consider the interaction of vertices and faces; the interaction of edges and faces must also be taken into account [6].

Acknowledgements

I would like to thank Mike Brady, John Hollerbach, Berthold Horn, Matt Mason and Patrick Winston for their suggestions on the contents and presentation of this paper. This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research is provided by the Office of Naval Research under contract N00014-77-C-0389.

References

[1] **Adamowicz, M.** The optimum two-dimensional allocation of irregular, multiple-connected shapes with linear, logical and geometric constraints, PhD Thesis, Department of Electrical Engineering, New York University, 1970.

[2] **Adamowicz, M and Albano, A.** "Nesting two-dimensional shapes in rectangular modules," *Computer Aided Design* **8**, 1 (Jan 1976), 27–32.

[3] **Albano, A and Sapuppo, G.** "Optimal Allocation of Two-Dimensional Irregular Shapes Using Heuristic Search Methods," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-10**, 5 (May 1980), 242–248.

[4] **Benson, R V.** *Euclidean Geometry and Convexity* , McGraw Hill, New York, 1966.

[5] **Bentley, J L and Ottman, T.** "Algorithms for Reporting and Counting Geometric Intersections," *IEEE Transactions on Computers* **C-28**, 9 (September 1979).

[6] **Boyse, J W.** "Interference Detection Among Solids and Surfaces," *Communications of the ACM* **22**, 1 (January 1979), 3–9.

[7] **Brown, K Q.** "Fast Intersection of Half Spaces," Carnegie-Mellon University, Department of Computer Science, CS-78-129, June 1978.

[8] **Eastman, C E.** "Preliminary report on a System for General Space Planning," *Communications of the ACM* **15**, 2 (February 1972), 76–87.

[9] **Eddy, W.** "A New Convex Hull Algorithm for Planar Sets," *ACM Transactions on Mathematical Software* **3**, 4 (December 1977), 398–403.

[10] **Freeman, H.** "On the Packing of Arbitrary-Shaped Templates," *Second USA-Japan Computer Conference* , 1975, 102–107.

[11] **Giralt, G; Sobek, R and Chatila, R.** "A Multilevel Planning and Navigation System for a Mobile Robot," *Sixth International Joint Conference on Artificial Intelligence* , Tokyo, Japan, August 1979, 335–338.

[12] **Graham R L.** "An Efficent Algorithm for Determining the Convex Hull of a Finite Planar Set," *Information Processing Letters* **1** (1972), 132–133.

[13] **Grunbaum, B.** *Convex Polytopes* , Wiley Interscience, New York, 1967.

[14] **Howden, W E.** "The Sofa Problem," *Computer Journal* **11**, 3 (November, 1968), 299–301.

[15] **Jarvis, R A.** "On the Identification of the Convex Hull of a Finite Set of Points in the Plane," *Information Processing Letters* **2** (1973), 18–21.

[16] **Larson, R C and Li, V O K.** "Finding Minimum Rectilinear Distance Paths in the Presence of Obstacles," MIT Operations Research Center, OR O88-79, May 1979.

[17] **Lewis, R A.** "Autonomous Manipulation on a Robot: Summary of Manipulator Software Functions," Jet Propulsion Laboratory, California Institute of Technology, TM 33-679, March 1974.

[18] **Lieberman, L and Wesley, M A.** "AUTOPASS: An Automatic Programming System for Computer Controlled Assembly," *IBM Journal of Research and Development* **21**, 4 (July 1977).

[19] **Lozano-Perez, T.** "The Design of a Mechanical Assembly System," MIT Artificial Intelligence Laboratory, TR-397, Dec 1976.

[20] **Lozano-Perez, T.** "Spatial Planning with Polyhedral Models," MIT Artificial Intelligence Laboratory, Technical Report, September 1980.

[21] **Lozano-Perez, T and Wesley, M A.** "An Algorithm for Planning Collision-Free Paths among Polyhedral Obstacles," *Communications of the ACM* **22**, 10 (October 1979), 560–570.

[22] **Lyusternik, L (translated from the Russian by Smith, T).** *Convex Figures and Polyhedra* , Dover Publications, U. K., 1963, Original Copyright Moscow 1956.

[23] **Moravec, H P.** "Visual Mapping by a Robot Rover," *Proceedings Sixth International Joint Conference on Artificial Intelligence* , Tokyo, Japan, August 1979.

[24] **Mueller, D and Preparata, F.** "Finding the Intersection of Two Convex Polyhedra," Coordinated Science Laboratory, Univ of Illinois - Urbana, R-793, Oct 1977.

[25] **Nilsson, N.** "A Mobile Automaton: An Application of Artificial Intelligence Techniques," *Proceedings International Joint Conference on Artificial Intelligence* , 1969, 509–520.

[26] **Paul R P.** "Manipulator Cartesian Path Control," *IEEE Transactions on Systems, Man, and Cybernetics* **SMC-9**, 11 (November 1979), 702–711.

[27] **Pfister, G.** "On Solving the FINDSPACE Problem, or How to Find Where Things Aren't," MIT Artificial Intelligence Laboratory. Working Paper 113, March 1973.

[28] **Pieper, D. L..** "The Kinematics of Manipulators under Computer Control," Stanford Artificial Intelligence Laboratory, AIM-72, Oct 1968.

[29] **Preparata, F and Hong, S.** "Convex Hulls of Finite Sets of Point in Two and Three Dimensions," *Communications of the ACM* **20**, 2 (Feb 1977), 87–93.

[30] **Preparata, F and Mueller, D.** "Finding the Intersection of a Set of Half Spaces in Time $O(n \log n)$," Coordinated Science Laboratory, Univ of Illinois - Urbana, R-793, Oct 1977.

[31] **Reif, J.** "On the Movers Problem," *20th Symposium on Foundation of Computer Science* , 1979, 421–427.

[32] **Schachter, B.** "Decomposition of Polygons into Convex Sets," *IEEE Transactions on Computers* **C-27**, 11 (November 1978), 1078–1082.

[33] **Shamos, M I.** "Geometric Complexity," *7th ACM SIGACT Symposium*, 1975, 224–233.

[34] **Shamos, M I and Hoey, D.** "Closest-Point Problems," *16th Symposium on Foundation of Computer Science*, 1975, 151–161.

[35] **Shamos, M I and Hoey, D.** "Geometric Intersection Problems," *17th Symposium on Foundation of Computer Science*, 1976, 208–215.

[36] **Stoyan, Y G and Ponomarenko L D.** "A Rational Arrangement of Geometric Bodies in Automated Design Problems," *Engineering Cybernetics* **16**, 1 (January 1978).

[37] **Sussman, G.** "The FINDSPACE Problem," MIT Artificial Intelligence Laboratory, Working Paper 18, Aug 1971.

[38] **Taylor, R.** "A Synthesis of Manipulator Control Programs from Task-Level Specifications," Stanford Artificial Intelligence Laboratory, AIM-282, July 1976.

[39] **Thompson, A M.** "The Navigation System of the JPL Robot," *Fifth International Joint Conference on Artificial Intelligence*, Massachusetts Institute of Technology, 1977.

[40] **Udupa, S.** "Collision Detection and Avoidance in Computer Controlled Manipulators," *Fifth International Joint Conference on Artificial Intelligence*, Massachusetts Institute of Technology, 1977.

[41] **Udupa, S.** Collision Detection and Avoidance in Computer Controlled Manipulators, PhD Thesis, Department of Electrical Engineering, California Institute of Technology, 1977.

[42] **Warnock, J E.** "A Hidden Line Algorithm for Halftone Picture Representation," Department of Computer Science, University of Utah, TR4-5, May 1968.

[43] **Whittaker, E T.** *Analytical Dynamics*, Dover, U. K., 1937.

[44] **Widdoes, C.** "A Heuristic Collision Avoider for the Stanford Robot Arm," Stanford Artificial Intelligence Laboratory, 1974.

[45] **Williams, J.** STICKS – a new approach to LSI design, SM Thesis, Department of Electrical Engineering, Massachusetts Institute of Technology, 1977.

[46] **Winograd, T.** *Understanding Natural Language*, Academic Press, New York, 1972.

[47] **Woo, T C.** "Progress in Shape Modelling," *Computer* **10**, 12 (December 1977).

Appendix 1. Algorithm for $CO_A^{xy}(B)$

This appendix shows an algorithm for $A \oplus B$, `SET-SUM(A, B)`, when A and B are convex polygons. Section 5 shows how this operation can be used to compute CO^{xy}.

Each polygon is described in terms of its vertices and the angles that the edges make with the positive x axis. The edges and vertices are ordered in counterclockwise order, i.e. by increasing angle. The polygon structure in the following program has the following components:

1. `size` — number of edges in polygon.

2. `vert [0:size]` — an array of vectors representing the coordinates of a vertex. The i^{th} edge, $i = 1, \ldots,$ `size`, has the endpoints `vert[i-1]` and `vert[i]`. Note that `vert[0]=vert[size]`.

3. `angle [0:size]` — the angle that the normal of an edge makes with the x axis, monotonically increasing. For convenience `angle[0]=angle[size]`.

The algorithm follows directly from Theorem 6, noting that in this representation of polygons, edges are represented by successive vertices.

```
SET-SUM (a, b)
      { c = new-polygon (a.size + b.size);   /* create new polygon of max size */
       ea = 1; eb = 1; vc = 0; ang = 0; offset = 0;
       do { ea = ea + 1 }
           until (a.angle[ea] >= b.angle[1]
              and a.angle[ea - 1] <= b.angle[1])
       c.vert[0] = a.vert[ea] + b.vert[0];
       do { vc = vc + 1;
           ang = offset + a.angle[ea];
           if (ang <= b.angle[eb])
               then if (ea > a.size)                /* handle wraparound */
                       then { offset = 2pi; ea = 1; }
                       else ea = ea + 1;
           if (ang >= b.angle[eb])
               then eb = eb + 1;
           c.vert[vc] = a.vert[ea] + b.vert[eb];
         }
       until (ea = a.size and eb = b.size);
       c.size = vc;
}
```

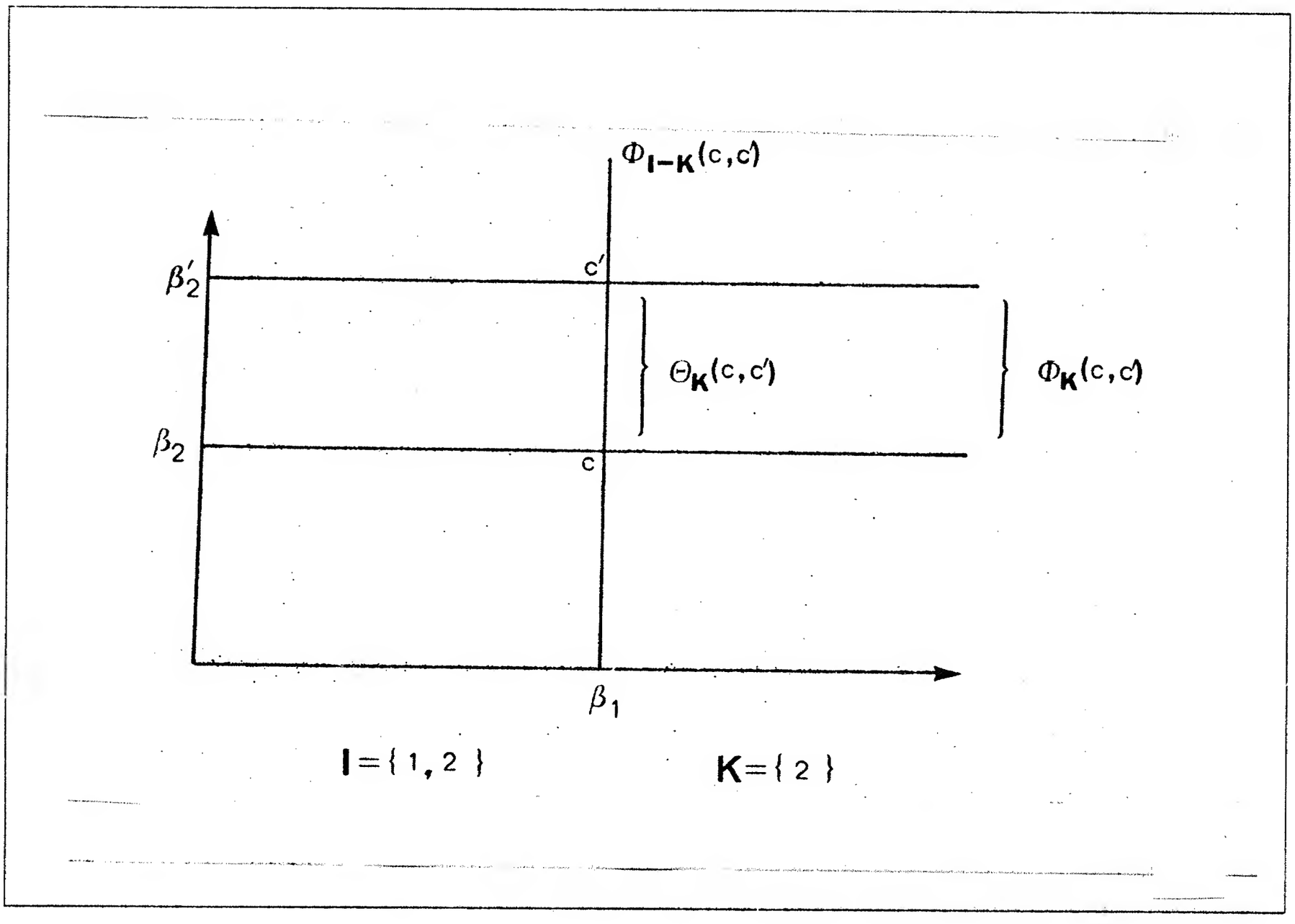


Figure A18. Illustration of the definition of $\Phi_K(c, c')$ and $\Theta_K(c, c')$.

Appendix 2. Proof of Theorem 8

Assume that $Cspace_A \subseteq \Re^d$, let $I = \{1, 2, \ldots, d\}$ and $K \subseteq I$. I, K and $I - K$ shall denote sets of indices for the coordinates of $a \in Cspace_A$. Define the following vectors, all in $Cspace_A$: $b = (\beta_i), c = (\gamma_i)$ and $c' = (\gamma'_i)$ for $i \in I$. Then,

$$\Phi_K(c, c') \equiv \{ b \in \Re^d \mid \bigwedge_{k \in K} \gamma_k \leq \beta_k \leq \gamma'_k \}$$

$$\Phi_K(c) \equiv \Phi_K(c, c)$$
$$\Theta_K(c, c') \equiv \Phi_K(c, c') \cap \Phi_{I-K}(c, c)$$

These definitions are illustrated in Figure A18.

The *projection operator*, denoted $P_K[\,\cdot\,]: \Re^d \mapsto \Re^{|K|}$ is defined, for vectors and sets of vectors, by:

$$P_K[b] = (\beta_k) \qquad k \in K$$
$$P_K[B] = \{ P_K[b] \mid b \in B \}$$

Superscripts on vectors indicate projection, e.g. $b^K = P_K[b]$. In addition, the vector in $\Re^{|I|}$ composed from one vector in $\Re^{|K|}$ and one in $\Re^{|I-K|}$ is denoted $(a^{I-K} : b^K)$, where $P_{I-K}[(a^{I-K} : b^K)] = a^{I-K}$ and $P_K[(a^{I-K} : b^K)] = b^K$.

In this notation, precise definitions for the notions of *cross-section projection* and *slice projection* can be provided. The cross-section projection of a $Cspace_A$ obstacle is written as follows:

$$CO_A(B)[c]_K \equiv P_{I-K}[CO_A(B) \cap \Phi_K(c)].$$

The slice projection, is similar to the cross-section projection, but carried out for all configurations between two cross-sections:

$$CO_A(B)[c, c']_K \equiv P_{I-K}[CO_A(B) \cap \Phi_K(c, c')].$$

The K-parameters of the two configurations, c and c', define the bounds of the slice. Similarly, the swept volume can be defined in this notation:

Definition: The *swept volume*[12] of A over the configuration range $[c, c']_K$ is

$$(A[c, c']_K)_c \equiv \bigcup_{a \in \Theta_K(c,c')} (A)_a$$

The requirement discussed in Section 10 that the swept volume of A be *displaceable* is embodied in the following condition:

$$\forall a: \bigcup_{x \in \Theta_K(c,c')} (A)_{(a^{I-K} : x^K)} = (A[c, c']_K)_{(a^{I-K} : c^K)} \qquad (4)$$

Note that the $I - K$ parameters may be changed, as in (4), but not those parameters in K. Therefore, $(A[c, c']_K)_a$ is defined only if $a \in \Phi_K(c)$.

Lemma 3: If (4) holds, i.e. if the swept volume is displaceable, then

$$P_{I-K}[CO_{A[c,c']_K}(B) \cap \Phi_K(c)] = P_{I-K}[(CO_A(B) \ominus \Theta_K(0, c' - c)) \cap \Phi_K(c)].$$

[12]The similarity in the notation between swept volume and slice projection does not imply any direct relationship.

Proof of Lemma:

($\supseteq$)

If $a \in P_{I-K}[\,(CO_A(B) \ominus \Theta_K(0, c'-c)) \cap \Phi_K(c)\,]$, then there exists an $(a^{I-K} : x_1^K) \in CO_A(B)$ and an $x_2 \in \Theta_K(0, c'-c)$ such that $x_1^K - x_2^K = c^K$. This implies that $x_1 \in \Phi_K(c, c')$. Then, using (4),

$$(A)_{(a^{I-K} : x_1^K)} \subseteq (A[c, c']_K)_{(a^{I-K} : c^K)}.$$

But since $(a^{I-K} : x_1^K) \in CO_A(B)$, then $(A)_{(a^{I-K} : x_1^K)}$ intersects B; therefore, its supersets also intersect B. By the definition of CO

$$(a^{I-K} : c^K) \in CO_{A[c,c']_K}(B) \cap \Phi_K(c) \Rightarrow a \in P_{I-K}[\,CO_{A[c,c']_K}(B) \cap \Phi_K(c)\,].$$

($\subseteq$)

Assume $a \in P_{I-K}[\,CO_{A[c,c']_K}(B) \cap \Phi_K(c)\,]$; then,

$$(A[c, c']_K)_{(a^{I-K} : c^K)} \cap B \neq \emptyset$$

For all $x \in \Theta_K(0, c'-c)$, $(a^{I-K} : c^K) + x \in \Phi_K(c, c')$ and $P_{I-K}[\,(a^{I-K} : c^K) + x\,] = a^{I-K}$. Therefore, by (4)

$$(A)_{(a^{I-K} : c^K)+x} \subseteq (A[c, c']_K)_{(a^{I-K} : c^K)}.$$

Since these are *all* the sets that make up the swept volume, at least one of them must also overlap B when the swept volume does; therefore there must be some $x_1 \in \Theta_K(0, c'-c)$ for which

$$(A)_{(a^{I-K} : c^K)+x_1} \cap B \neq \emptyset.$$

By the definition of CO,

$$(a^{I-K} : c^K) + x_1 \in CO_A(B) \Leftrightarrow (a^{I-K} : c^K) \in (CO_A(B) - \Theta_K(0, c'-c)) \cap \Phi_K(c).$$

Clearly, then

$$a \in P_{I-K}[\,(CO_A(B) - \Theta_K(0, c'-c)) \cap \Phi_K(c)\,]$$

∎

This Lemma leads to a proof of Theorem 8.

Theorem 8: If (4) holds, then

$$P_{I-K}[\,CO_A(B) \cap \Phi_K(c, c')\,] = P_{I-K}[\,CO_{A[c,c']_K}(B) \cap \Phi_K(c)\,]$$

Proof of Theorem:

The proof below shows that

$$P_{I-K}[\,CO_A(B) \cap \Phi_K(c, c')\,] = P_{I-K}[\,(CO_A(B) \ominus \Theta_K(0, c' - c)) \cap \Phi_K(c)\,].$$

The theorem follows directly from this result and Lemma 3.

($\subseteq$)

If $a \in P_{I-K}[\,CO_A(B) \cap \Phi_K(c, c')\,]$, then, for some $x \in \Theta_K(0, c' - c)$ there is an a_1 such that

$$a_1 = (a^{I-K} : c^K + x^K) \in CO_A(B) \cap \Phi_K(c, c')$$

Since $x^{I-K} = 0$, then $a_1 - x \in \Phi_K(c)$ and therefore

$$a_1 - x \in (CO_A(B) \ominus \Theta_K(0, c' - c)) \cap \Phi_K(c).$$

Since $a = P_{I-K}[a_1 - x]$

$$a \in P_{I-K}[\,(CO_A(B) \ominus \Theta_K(0, c' - c)) \cap \Phi_K(c)\,].$$

($\supseteq$)

Assume $a \in P_{I-K}[\,(CO_A(B) \ominus \Theta_K(0, c' - c)) \cap \Phi_K(c)\,]$, then,

$$(a^{I-K} : c^K) \in (CO_A(B) \ominus \Theta_K(0, c' - c)) \cap \Phi_K(c).$$

Since the $I - K$ parameters are not changed by the set subtraction, there must be an $a_1 = (a^{I-K} : x_1^K) \in CO_A(B)$. x_1 must be in $\Phi_K(c, c')$, because $P_K[x_1 - x_2] = c$ with $x_2 \in \Theta_K(0, c' - c)$.

$$(a^{I-K} : x_1^K) \in CO_A(B) \cap \Phi_K(c, c') \Rightarrow a \in P_{I-K}[\,CO_A(B) \cap \Phi_K(c, c')\,].$$

∎